UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TREBALL FI DE GRAU

Grau en Enginyeria Elèctrica

DISSENY I DESENVOLUPAMENT D'UN VIDEOJOC 3D MULTIJUGADOR

Memòria i Annexos

Autor: Eduard Sanfelix Morera
Director: Adriana Farran Marsà
Tutor: Samir Kanaan Izquierdo

Convocatòria: 06/2020

Resum

La indústria del videojoc es troba en un constant creixement, fins al punt de convertir-se en un dels sectors que ofereix majors oportunitats de treball i millors salaris. L'ocupació que genera el sector dels videojocs és molt amplia degut a que engloba multitud de disciplines. La llibertat dels desenvolupadors dona lloc a enfocar els videojocs segons la voluntat del creador, és a dir, permet enfocar un joc com una construcció cultural, o bé com un sistema narratiu i simbòlic. *Viking's Brothers* és un videojoc enfocat a una experiència audiovisual dissenyat per a tot tipus de públic. La transversalitat de la disciplina ha permès que aquest projecte sigui tant o més nutritiu per a la meva trajectòria acadèmica que un projecte relacionat amb el grau en curs. El procés de recerca d'informació i d'autoaprenentatge han estat determinants per obtenir la capacitat de dissenyar i produir el videojoc que s'explicarà en la present memòria.

Resumen

La industria del videojuego se encuentra en un constante crecimiento, hasta el punto de convertirse en uno de los sectores que ofrece mayores oportunidades de trabajo y mejores salarios. El empleo que genera el sector de los videojuegos es muy amplio debido a que engloba multitud de disciplinas. La libertad de los desarrolladores da lugar a enfocar los videojuegos según la voluntad del creador, es decir, permite enfocar un juego como una construcción cultural, o bien como un sistema narrativo y simbólico. *Viking's Brothers* es un videojuego enfocado a una experiencia audiovisual diseñado para todo tipo de público. La transversalidad de la disciplina ha permitido que este proyecto sea más nutritivo para mi trayectoria académica que un proyecto relacionado con grado en curso. El proceso de búsqueda de información y de autoaprendizaje han sido determinantes para obtener la capacidad de diseñar y producir el videojuego que se explicará en la presente memoria.

Abstract

The video game industry is constantly growing, to the point of becoming one of the sectors that offers greater job opportunities and better wages. The employment generated by the video game sector is very wide because it encompasses a multitude of disciplines. The freedom of developers gives rise to focus on video games according to the will of the creator, ie, allows you to focus a game as a cultural construction, or as a narrative and symbolic system. Viking's Brothers is a video game focused on an audiovisual experience designed for all types of audiences. The transversality of the discipline has allowed this project to be as much or more nutritious for my academic career than a project related to an ongoing degree. The process of information seeking and self-learning have been instrumental in gaining the ability to design and produce the video game that will be explained in this report.

Agraïments

És una bona oportunitat per agrair a totes les persones que han contribuït en la realització d'aquest projecte. En primer lloc m'agradaria agrair el suport rebut per part de la meva família, els meus pares i la meva germana. Als meus pares, que sempre m'han donat suport en totes les decisions que he pres durant la carrera i que sempre han sabut dona una dosi d'autoestima i positivisme en els moments més difícils. Sobretot al meu pare, que sempre ha tingut la voluntat d'ajudar-me donant el millor d'ell mateix. Voldria donar un especial reconeixement al professor i tutor del meu projecte, que m'ha donat la motivació quan més ho necessitava. També m'ha donat consells i recomanacions contribuint ha desenvolupar un projecte digne i complet.

Glossari

Online	*Alguna cosa o algú que està connectat fent ús d'una xarxa generalment d'internet.*
Software	*Programa o conjunt de programes que permeten realitzar tasques en un sistema informàtic.*
Asset Store	*Conté una biblioteca d'actius gratuïts i comercials creats per Unity Technologies i membres de la comunitat.*
Asset	*Recurs informàtic que conté components o funcionalitats d'un sistema d'informació.*
GameObject	*Són els objectes fonamentals d'Unity que representen personatges, accessoris i decorats.*
Script	*Document que conté instruccions escrites amb codi de programació.*
Multijugador	*Modalitat d'un videojoc que permet jugar a diversos jugadors simultàniament en una partida.*
Servidor	*Equip informàtic que forma part d'una xarxa i proveeix servei a altres equips.*
Avatar	*Imatge gràfica que representa a un humanoide utilitzat en mons virtuals.*
App	*Abreviatura de la paraula amb anglès application. És un software que s'instal·la per ajudar al usuari en una labor concreta.*
Tag	*Etiqueta que s'utilitza coma paraula clau en els llenguatges informàtics.*
Trigger	*Paràmetre d'un collider per convertir-lo en un disparador.*
Input	*Conjunt de dades que s'introdueixen en un sistema o programa informàtic*
Target	*Objectiu, persona o alguna cosa a la que es dirigeix una acció.*
Prefab	*Objectes reutilitzables creats amb una sèrie de característiques, dins el llenguatge propi d'Unity.*
Stamina	*Resistència que es té sobre una acció.*

Índex

Índex de figures

Índex de taules

1. Prefaci

L'existència de continguts, objectius i competències simultànies en una mateixa matèria ha estat determinant alhora de emprendre el desenvolupament d'aquest projecte i de trobar la motivació per tirar-lo endavant. La realització de treballs en cursos anteriors relacionats amb la robòtica i programació mostra l'interès existent en aquesta matèria i la voluntat per seguir aprenent altres aspectes relacionat amb aquest àmbit, com la realització i producció d'un videojoc multijugador amb plataformes 3D.

1.1. Origen del treball

Arran de cursar l'assignatura de Programació de dispositius mòbils, el professor i coordinador de la assignatura va donar l'opció de desenvolupar una aplicació o bé un videojoc com a treball de fi de grau. En primera instància em va semblar inapropiat per la meva trajectòria acadèmica, però el desenvolupament del projecte en l'assignatura em va fer veure la transversalitat de la matèria i vaig emprendre el treball de fi de grau com una oportunitat per ampliar els coneixements en àmbits que en ningun cas podria obtenir en el grau d'enginyeria elèctrica.

1.2. Requeriments previs

La formació rebuda durant el transcurs de l'assignatura va ser clau per adquirir les capacitats necessàries per seguir amb la formació del software i per aprendre el llenguatge de programació que utilitza. El coneixement bàsic del llenguatge ha sigut determinant per seguir amb el procés d'aprenentatge, ja que iniciar un projecte d'aquesta envergadura sense tenir coneixements previs és gairebé impossible de ser desenvolupat amb èxit.

El grau en enginyeria elèctrica ha resultat imprescindible en la recerca d'informació útil i en la capacitat de solucionar una gran quantitat de dificultats que s'han presentat al llarg d'aquest projecte. Alhora de cercar la informació necessària per dur a terme el projecte, la capacitat d'anàlisis atorgada durant la meva formació a la universitat ha estat clau per obtenir dades i tractar-les en funció de la situació i dels problemes que es presentaven. El disseny modular ha estat determinant per manipular independentment cada element del videojoc i combinar-los per formar el disseny final del projecte.

1.3. Motivació

Durant l'assignatura es va dissenyar, juntament amb el meu equip de treball, un videojoc 2D com a projecte final de la matèria. El resultat d'aquest va ser molt positiu i va donar lloc a la motivació per desenvolupar una ampliació d'aquest o bé dissenyar-ne un de nou. El fet de modificar i millorar el projecte ja existent, acotava les possibilitats i limitava el procés de creació. Per tant, es va decidir dissenyar un videojoc sense cap restricció amb la premissa de convertir-lo amb un projecte desenvolupat amb tres dimensions incorporant la capacitat de connexió *online* amb més jugadors.

2. Introducció

El disseny i producció d'un videojoc és el procés de creació d'un projecte audiovisual on una o més persones interaccionen per mitjà d'un controlador. Aquest procés es pot dur a terme per una gran empresa, un petit equip de desenvolupadors, o fins i tot per un sol individu. Desenvolupar un videojoc amb una gran empresa ofereix la possibilitat de dissenyar un projecte complet i d'alta complexitat, en canvi, el desenvolupador independent haurà de limitar el seu videojoc segons les eines i capacitats del mateix. No obstant, té la gran avantatja de permetre dissenyar el projecte enfocat als seus interessos i motivacions. En les grans empreses, el dissenyador és una persona més dins l'equip que treballa en funció dels gustos i interessos de l'empresa, limitant la seva creativitat segons les condicions establertes per tot l'equip.

Es distingiran tres grans blocs que defineixen l'estructura del projecte. En primer lloc, s'explicaran els aspectes i característiques de la fase de predissey i disseny del videojoc. En aquest bloc s'acotaran els límits del projecte i es decidirà l'estructura de treball, definint la metodologia i establint les eines necessàries per dur a terme el videojoc. El segon bloc es dedicarà a l'anàlisi dels elements dissenyats per confeccionar el videojoc. Es detallaran els components que formen l'estructura i es mostrarà la implementació d'aquests en el software escollit en la fase de predisseny. Finalment es proposaran possibles millores que es podrien afegir al videojoc, així com formes d'optimització de recursos amb la finalitat de crear una versió millorada del projecte.

S'han establert una sèrie d'objectius que marcaran la metodologia en la qual es formarà el projecte. Els objectius generals i específics donaran lloc a definir l'abast del projecte, és a dir, fins on es vol arribar i quines seran les tasques a desenvolupar durant el transcurs d'aquest treball. La finalitat gira entorn a la creació d'un videojoc multijugador dissenyat amb models 3D. La idea inicial és dissenyar un videojoc complet amb totes les funcionalitats implementades per tal d'oferir-lo a la comunitat de videojocs. El resultat final serà l'única variable de decisió que determinarà si el videojoc és apte per a ser compartit amb la resta de desenvolupadors i jugadors.

2.1. Objectius del treball

Els objectius d'aquest treball donen lloc a plantejar una planificació inicial de la metodologia de treball. Marcaran els passos a seguir que guiaran les tasques a realitzar de forma ordenada i amb la finalitat d'assegurar si el projecte es viable i es podrà desenvolupar amb èxit. A continuació es detallen els objectius generals i específics que assentaran les bases per dur a terme un videojoc exitós.

- Definició de les característiques del videojoc.
    - Realitzar un estudi dels possibles gèneres que pot adoptar un videojoc.
    - Adaptació del videojoc en funció del gènere escollit.
- Implementació del disseny modular per desenvolupar el projecte.
    - Investigació dels motors disponibles per desenvolupar un videojoc.
    - Elaborar des de zero el disseny dels personatges, escenaris i funcionalitats.
- Implementació d'un sistema multijugador.
- Desenvolupar un videojoc complet per poder ser compartit amb la comunitat.
    - Implementar les millores necessàries per fer més atractiu el videojoc.
    - Solucionar falles o defectes per millorar la robustesa del videojoc.

El criteri de selecció dels objectius es basa en especificar unes fites que siguin mesurables i realistes, és a dir, tenint en compte els recursos disponibles s'estableixen uns objectius exigents però assolibles. Aquests objectius estan limitats per les condicions i capacitats d'un mateix i pel temps disponible.

2.2. Abast del treball

Aquest projecte abastirà totes les tasques relacionades amb el disseny i creació d'un videojoc multijugador. Es crearan uns models 3D que s'utilitzaran com a personatges principals, incorporant les funcions i animacions per convertir-lo en un model completament funcional. Es dissenyaran una sèrie de personatges antagònics amb les funcions i animacions corresponents. Les animacions que s'afegiran als models objecte d'aquest projecte. Es buscarà la forma d'afegir-les a partir d'un *software* extern que ho faci possible.

Es crearà un escenari partint d'un esbós basat amb la experiència obtinguda com a consumidor de videojocs. Es dissenyaran objectes i funcionalitats que relacionen els personatges principals amb els enemics i que donaran la jugabilitat necessària per aconseguir l'entreteniment que requereix un videojoc. No es dedicaran esforços amb el disseny estètic dels enemics i objectes que puguin aparèixer a l'escena. Els models 3D seran obtinguts a partir d'altres fonts, es centrarà amb les tasques de programació per afegir funcionalitats i característiques a aquests models.

S'implementarà un sistema multijugador que permetrà connectar jugadors simultàniament en una partida. Els servidors que permeten la connexió *online* seran subministrats per alguna plataforma que ofereixi aquest servei. Es centrarà en la programació d'aquest sistema per fer-la completament funcional.

3. Disseny del videojoc

El procés de creació d'un videojoc és una activitat multidisciplinària que transforma un concepte inicial fins arribar al producte final. Requereix coneixements de diferents camps per la gran quantitat d'aportacions creatives necessàries per desenvolupar un videojoc.

Per al predisseny d'aquest projecte, s'han seguit els passos que es descriuen a continuació:

1. Preconcepció de la idea general del videojoc, sobre els tipus de personatges, ambientació, escenari...
2. Definir el gènere i estil del projecte.
3. Definir l'estil de la càmera, i la perspectiva que adoptarà respecte el jugador.
4. Definir l'argument que seguirà el videojoc, és a dir, donar sentit a les accions que s'estan realitzant.
5. Marcar uns objectius a assolir segons les intencions del dissenyador, on es vol arribar i de quina manera.
6. Tenir clar quin serà o seran els components que generaran diversió al jugador.
7. Definir les regles del joc per establir un equilibri entre tots els components del sistema (*game balance*).
8. Escollir quin motor s'utilitzarà per dissenyar el videojoc segons els requeriments de l'usuari.
9. Definir les funcionalitats que tindrà el personatge principal, l'entorn i el conjunt del videojoc.

Durant la fase de disseny, s'han de detallar tots els elements que compondran el joc, donant una primera idea de les peces que el formaran. Per relacionar aquests elements es defineix una **història** que relacioni el desenvolupament dels personatges i l'ambientació del món representat. Es definirà el context de la història, es determinaran els objectius del joc, com seran els personatges principals i quina relació tindran amb l'escenari i la resta de personatges, donant lloc a un primer disseny del **guió** del joc.

Un cop definida la història i el guió del joc s'estableix l'**aspecte general** que es vol aconseguir i una conceptualització del disseny dels personatges, escenari, objectes i enemics. També es decidirà els **elements sonors**, ja siguin sons ambientals, efectes de so o les veus dels personatges. Llavors s'especificarà el funcionament de tot el conjunt i les mecàniques del joc, en funció del gènere i l'estil escollits en la fase de predisseny.

Després s'inicia el procés de recerca de recursos segons la conceptualització definida en la fase de disseny. Per fer-ho, s'ha fet ús dels recursos que ofereix *l'Asset Store* anomenats *assets.* També s'han

utilitzats altres softwares com *Fuse* per dissenyar els personatges principals o *Mixamo* per a la incorporació d'animacions.

La fase d'implementació dels elements al motor és la tasca més costosa d'efectuar ja que engloba el control d'aquests i com es desenvolupen amb la resta d'elements del videojoc.

El disseny de la interfície gràfica de l'usuari (UI) s'ha afegit un cop s'ha disposat de tots els elements. Consta d'una fase de predisseny on es contemplen les possibles accions que desenvoluparan, així com l'aspecte visual que tindran.

Finalment s'ha incorporat el mode multijugador amb la creació d'un escenari i un sistema de control similar però independent al mode principal. Es va iniciar un procés de recerca per determinar l'eina més adient per implementar la connexió online entre jugadors i s'han dissenyat elements de la UI exclusius per a aquest mode de joc.

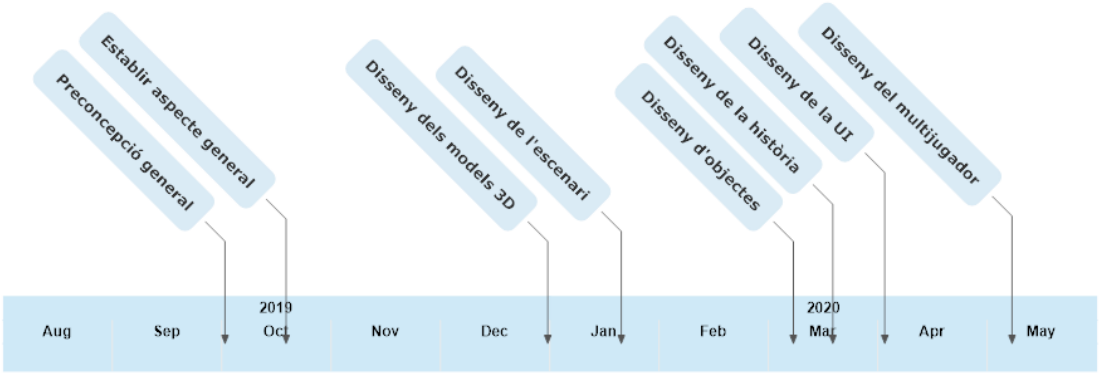


Figura 3.1 *- Fites del projecte.*

La manca d'experiència en aquest tipus de projectes ha conduït a construir el videojoc a partir d'un **model iteratiu**. El procés es basa en afegir complexitat de forma progressiva fins arribar al disseny final. A diferència d'un procés seqüencial, on es desenvolupa estrictament cada etapa pas a pas, el model iteratiu repeteix un conjunt petit d'etapes donant lloc a una cicle. Aquest cicle, estipulat en la fase de planificació inicial, s'itera indefinidament amb l'objectiu de millorar gradualment projecte. En cada iteració s'afegeix les millores aconseguides en la iteració anterior, per tal de millorar la iteració següent. El model s'empren amb una planificació inicial que donarà lloc al procés iteratiu. El procés es repetirà un nombre indefinit de vegades fins arribar al desplegament final del projecte, i consta de les següents etapes:

- **Planificació:** etapa inicial que recull els conceptes estipulats en la fase de predisseny.
- **Requisits:** establir quins elements es necessitaran per dur a terme les pròximes etapes del cicle.
- **Anàlisi i disseny:** es realitza un anàlisi de la lògica que es té que aplicar en les següents fases, així com les dades que es requeriran per desenvolupar el projecte. En l'etapa de disseny s'estableixen els requisits tècnics per satisfer les necessitats de l'etapa d'anàlisi.
- **Implementació:** s'inicia el procés de codificació dels elements i aspectes definits a les etapes anteriors.
- **Probes:** consta d'un procés per identificar els errors o problemes que hagin pogut sorgir en l'etapa d'implementació.
- **Avaluació:** un cop completades les etapes anteriors, es valoren els resultats obtinguts i s'avalua el desenvolupament complet fins aquesta etapa.

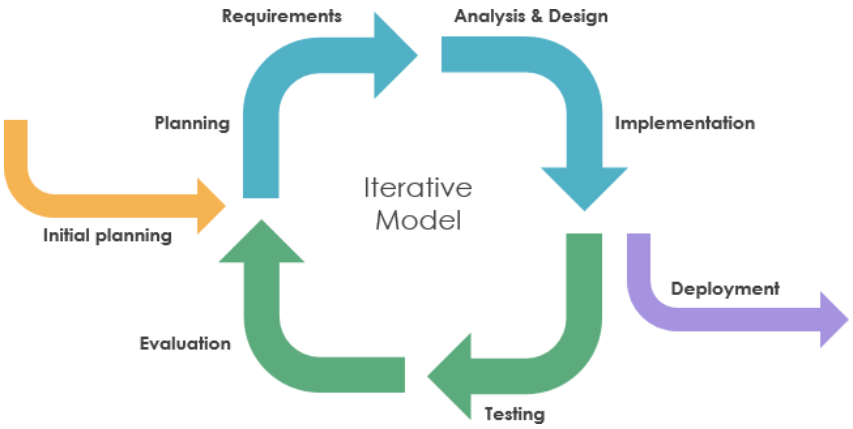


***Figura 3.2**- Esquema de les etapes del model iteratiu.*

3.1. Gènere

Per distingir un videojoc d'un altre es va idear una forma de classificar-los basat en la seva pròpia jugabilitat sense tenir en compte l'aspecte gràfic i visual. El gènere és una forma de reconèixer un videojoc segons l'estil i les mecàniques del joc independentment de l'ambientació.

Al 1984, el físic i dissenyador de videojocs Chris Crawford va fer la primera distinció de gèneres de videojocs classificant-los en dos grans pilars: jocs d'acció i jocs d'estratègia. A dia d'avui, la complexitat dels videojocs a incrementat exponencialment amb els anys forçant a la comunitat a afegir subcategories basades en les mecàniques del joc. El tipus d'interacció del jugador amb la màquina i el sistema de joc són uns dels factors principals que determinen el gènere d'un videojoc. Cada cop és més normal que els videojocs continguin elements de diferents gèneres forçant a que un mateix joc tingui més d'un estil que el caracteritza.

Aquest projecte es centrarà amb els **videojocs de rol** o també anomenats amb sigles angleses com **RPG** (role-playing game). Aquest gènere es caracteritza per la interacció amb el personatge, una història profunda y una evolució d'aquest a mesura que es progressa amb la història. L'evolució del personatge s'assoleix amb l'exploració de l'escenari aconseguint noves armes, noves habilitats i el coneixement progressiu dels elements de la història. La narrativa, ja sigui amb quadres de diàleg o amb cinemàtiques, és un element imprescindible que contribueix a la immersió de la història i millora l'experiència del jugador.

Segons la finalitat i la configuració del videojoc, es destaquen els següents subgèneres:

- **Videojoc de rol pur:** és l'estil clàssic, inspirats amb els jocs de taula on cada jugador utilitza les habilitats apreses durant el transcurs de la història realitzant batalles per torns amb els enemics.
- **Videojoc de rol-acció:** comparteix gairebé totes les característiques amb el gènere original però ofereix un estil de combat diferent. El sistema de batalla entre el jugador i els enemics és a temps real. És el subgènere que comparteix més característiques amb el present videojoc *Viking's Brothers.*
- **Videojoc de rol multijugador massiu:** s'utilitza en videojocs multijugador que permeten interactuar entre un gran nombre de jugadors en línia dins un món virtual extens.
- **Videojoc de rol tàctic:** aquest subgènere conté elements de videojocs d'estratègia combinats amb les característiques d'un videojoc de rol original.

3.2. Càmera

La càmera en un videojoc és l'element que proporciona la vista del món virtual i la perspectiva del personatge principal. És un dels elements que defineix l'estil i modifica la jugabilitat del conjunt. Existeixen diferents perspectives de càmera en funció de l'efecte que es vol aconseguir i de la sensació que es vulgui transmetre a l'usuari. Les més utilitzades en la majoria de videojocs són:

- **Primera persona:** aquesta vista permet veure el món virtual des de la perspectiva del personatge principal. S'utilitza aquest tipus de càmera quan es vol transmetre realisme i una sensació de presencia dins l'escenari. No obstant, la perspectiva no permet veure amb facilitat els elements que rodegen el mapa, limitat per la vista del jugador.
- **Tercera persona:** té com a característica que el personatge principal es veu de cos sencer i d'espatlles. La finalitat d'aquesta vista és ampliar el camp de visió del jugador, facilitant la recerca d'objectes i elements que es troben dins el món virtual. A diferència de la primera persona, aquesta permet veure els voltants sense necessitat de girar la vista del jugador.

En aquest projecte s'ha optat per incorporar la càmera en tercera persona ja que és la perspectiva que més s'adapta als requeriments del videojoc.

Figura 3.3 *- Perspectiva de la càmera en tercera persona.*

3.3. Argument

La jugabilitat i els gràfics són paràmetres que determinen la qualitat d'un videojoc. No obstant, quan parlem de l'estil RPG és imprescindible tenir un argument sòlid que ajudi en la immersió de la història.

Una història ben construïda s'aconsegueix buscant generar un efecte de curiositat i interès per donar sentit al jugador per seguir progressant. S'ha de crear una ambientació acord amb l'entorn i una sèrie de fites que motivin al jugador per avançar fins arribar al següent objectiu.

Viking's Brothers està ambientat en un poblat víking situat en un terreny muntanyós. Tracta d'un noi anomenat Harald que retorna a la seva aldea després de molt de temps i es troba el poblat en runes. A mesura que avança amb el relat s'adona de l'aparició d'una maledicció que ha provocat la eliminació dels habitants del seu poble. Harald descobrirà que la maledicció es pot vèncer si aconsegueix trobar uns tresors que el conduiran fins a l'enemic final. Els tresors estaran repartits per l'escenari en zones on es trobaran enemics que defensaran aquest objecte. El jugador haurà de investigar com arribar fins als tresors superant proves i enigmes dissenyats per afegir dificultat i curiositat a la història.

3.4. Objectiu del joc

L'argument del joc dona pas a determinar un objectiu pel qual el jugador trobarà la motivació per avançar amb el nivell. S'han establert una sèrie d'objectius que s'aniran assolint progressivament segons les indicacions del sistema de diàlegs.

1. El jugador haurà de buscar una arma principal per poder derrotar als enemics. S'indicarà també la necessitat d'obtenir un atac especial per poder superar els següents objectius.
2. S'aconsellarà al jugador de la importància de trobar uns tresors amagats en punts estratègics de l'escenari. A mesura que es superen els primers obstacles, s'explicarà al jugador el motiu de la importància d'aquests tresors.
3. Es destacarà la necessitat de trobar unes claus que donen pas a obtenir un dels tresors amagats. Per assabentar d'aquest fet al jugador, s'indicarà amb un diàleg la importància d'aquests objectes.
4. S'explicarà al jugador la forma de superar el nivell quan es trobi en un punt avançat de la història. Es presentarà l'enemic final i se l'indicarà com trobar-se amb ell. El jugador haurà d'esbrinar com derrotar-lo ja que aquest presenta unes condicions diferents a la resta.
5. Finalment, s'accedirà a un objecte que el jugador haurà d'interaccionar amb ell per superar el nivell complet.

És imprescindible assolir tots els objectius de manera progressiva, en cas contrari, el jugador es veurà obligat a tirar enrere i buscar la forma de completar els objectius que se li encomanen.

4. Motor de videojoc

Un motor de videojoc (*game engine)* és un conjunt d'eines que ajuden a realitzar un procés de creació d'un videojoc. Els motors donen eines al programador per desenvolupar la idea general del videojoc i així, evitar invertir temps en tasques que et facilita el propi motor. Fan referència a una sèrie de llibreries de programació que permeten dissenyar, crear i representar un videojoc.

Les funcionalitats bàsiques d'un *game enigne* inclou:

- **Motor gràfic:** permet renderitzar gràfics amb 2D i 3D, és a dir, generar una representació gràfica d'un model bidimensional o tridimensional.
- **Motor físic:** eina que permet implementar les lleis de la física i simular-les dins la mecànica del joc. Permet tenir una sensació més realista alhora d'interaccionar amb els objectes de l'escena. També incorpora un detector de col·lisions per simular la resposta d'un objecte davant un estímul i donar la capacitat al programador de reaccionar quan es produeix una col·lisió.
- **Motor d'animacions:** eina que permet crear imatges en moviment i modelar-les dins l'escena del videojoc.
- **Motor de sons:** encarregat de processar pistes de so, modificar-les i reproduir-les dins l'escena quan el programador ho trobi oportú.
- **Scripting:** permet al desenvolupador prendre el control de l'escena, manipular-la i implementar el funcionament dels personatges i objectes que formen part del videojoc.
- **Inteligència artificial:** l'eina permet crear comportaments programats i dissenyar una presa de decisions per aconseguir una consistència lògica i coherent.
- **Gestió de memòria:** tots els components del videojoc consumeixen recursos. L'eina permet optimitzar alguns components per reduir el consum de memòria i millorar la fluïdesa del conjunt.

Per decidir quin motor és el més adient per al desenvolupament d'un videojoc s'han de tenir en compte les capacitats gràfiques, ja que són les encarregades de mostrar les imatges 2D o 3D a la pantalla i de realitzar càlculs d'aspectes d'il·luminació, textura... Un altre aspecte a l'hora d'escollir un motor és la facilitat d'accedir als recursos i ajudes que ofereix la comunitat o el propi software, degut a la necessitat d'aprendre constantment diferents matèries de qualsevol disciplina.

4.1. Elecció del motor

Tot hi haver treballat prèviament en assignatures del grau amb Unity3D, s'ha realitzat un estudi previ per determinar el motor que s'ajusti millor a les característiques escollides en el predisseny del videojoc.

Els motors més utilitzats per la comunitat són Unity 3D, Unreal Engine i Godot. Tots tres tenen les seves avantatges i inconvenients segons el tipus de videojoc que es vulgui dissenyar. Per això, s'han estudiat les característiques de cadascun d'ells i s'escollirà el més adient en funció dels requeriments estipulats en la fase de predisseny.

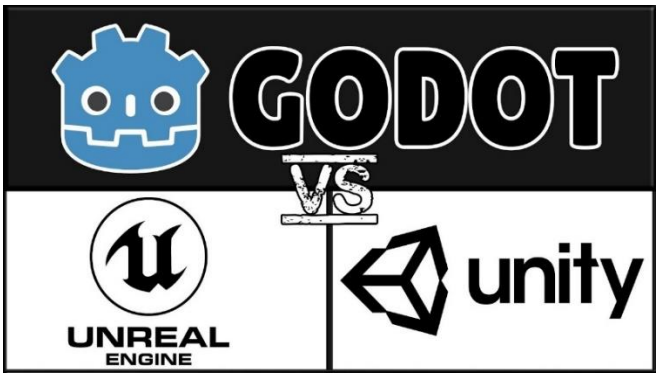


Figura 4.1 *– Motors de videojocs més utilitzats.*

UNITY 3D

És l'eina més famosa i utilitzada per la comunitat. Unity ofereix tres alternatives de subscripció basades amb els ingressos del videojoc, és a dir, si l'empresa (o el petit desenvolupador) ingressa menys de 100.000$ anuals es permetrà utilitzar la llicència personal gratuïta. Si els ingressos es situen entre els 100.000$ - 200.000$ anuals, s'optarà per una llicència plus que són 20€ mensuals. Si es sobrepassa dels 200.000$ anuals, s'haurà de contractar una llicència superior que són 115€ mensuals.

La documentació que ofereix Unity 3D és molt amplia. Dins el seu propi manual hi ha exemples de codi, exemples d'ús i guies d'utilització de les eines implementades al software. La disposició del manual és molt intuïtiva i fàcil de utilitzar.

Una de les grans avantatges d'aquest motor són la gran quantitat de plataformes que pot suportar. Es compatible amb més de 25 plataformes, engloba des de la realitat virtual (VR) fins totes les possibles consoles que existeixen en el mercat actual.

En quan al llenguatge de programació, Unity 3D utilitza el C# per implementar codi al videojoc. Aquest llenguatge va ser creat per l'empresa Microsoft i posteriorment es va estandarditzar. Per poder-lo utilitzar es requereix una formació prèvia ja que es tracta d'un llenguatge complex. Unity 3D ofereix la possibilitat de incorporar eines de programació visual, però no estan incloses per defecte. També utilitza un llenguatge dissenyat específicament per a l'ús de Unity anomenat UnityScript.

La forma d'utilitzar les eines de Unity 3D i d'interactuar amb la seva interfície fa que la corba d'aprenentatge sigui baixa.

Per afegir recursos addicionals, Unity 3D disposa de l'Asset Store per descarregar models 3D, textures o eines necessàries per desenvolupar un videojoc.

UNREAL ENGINE

És un dels motors més veterans que va sorgir al voltant de l'any 1999 amb l'estrena del famós videojoc *Unreal*. Les subscripcions d'Unreal Engine es basen amb un sistema de *royalties* (quantitat a pagar per l'ús d'una patent), la qual no es tindrà que pagar cap import fins arribar a ingressar 3000$ per trimestre. A partir d'aquesta xifra, es tindrà que pagar a Unreal Engine el 5% dels ingressos.

La documentació d'Unreal Engine és excel·lent però l'estructura del manual és menys intuïtiva. Hi ha gran quantitat d'informació però la complexitat de la interfície dificulta la recerca d'aquesta informació.

Comparat amb Unity 3D, aquest motor només suporta 18 plataformes diferents. Aquesta dada pot arribar a ser molt rellevant si el videojoc que es vol dissenyar està pensat per alguna plataforma menys habitual.

Unreal Engine té l'opció de treballar amb dos llenguatges de programació diferents, C++ i Blueprints. Aquest últim és un llenguatge més visual pensat per a programadors novells propi d'Unreal Engine.

Tot i la informació que ofereix el manual d'Unreal Engine sobre les funcions del seu software, la corba d'aprenentatge és més pronunciada que en el cas de Unity 3D.

En el cas de Unreal Engine, disposa d'un banc de recursos anomenat *Marketplace.* És molt similar al que utilitza Unity 3D ja que ofereix el mateix tipus d'elements però la interfície gràfica és diferent.

GODOT

El major atractiu que ofereix aquest motor és la subscripció lliure, és a dir, no es té que pagar en funció del que es genera.

La informació que disposa el manual de Godot és molt amplia i la interfície que presenta està molt ben organitzada i és molt intuïtiva.

La major limitació que té aquest software és el nombre de plataformes a les que es pot exportar el projecte que, en aquest cas, només són 6. Està dissenyat per desenvolupar videojocs en dispositius mòbils o en sistemes operatius d'ordenador.

La gran avantatja de Godot és el nombre de llenguatges de programació que suporta. Té un llenguatge de programació propi anomenat *GDScript* que es caracteritza per la facilitat de manipular –lo. També te suport per a programació visual dedicat a usuaris amb menys experiència. En les últimes actualitzacions, Godot ha afegit la possibilitat de utilitzar els llenguatges de programació propis d'Unity 3D i d'Unreal, C# i C++.

La corba d'aprenentatge es considera molt similar a la d'Unity 3D, essent aquesta de grau mitg o baix.

La disponibilitat de recursos per a Godot és baixa, ja que no disposa d'un banc de recursos propi i es tenen que descarregar d'altres llocs web.

Finalment es presenta una taula resum amb les principals característiques dels motors explicats en aquest apartat.

	Unity 3D	**Unreal Engine**	**Godot**
Preu	0€/20€/ 115€ al mes	5% dels ingressos (a partir dels 3000$ per trimestre)	Gratuït
Plataformes d'exportació	+25	18	6
Documentació	3/3	3/3	3/3
Llenguatge de programació	C# / UnityScript	C++ / Blueprints	GDScript, Visual Scripting, C# i C++
Corba d'aprenentatge	Baixa/Mitja	Mitja/Alta	Baixa/Mitja
Recursos addicionals	Asset Store	Marketplace	No

Taula 4.1.- *Taula resum dels motors de videojocs.*

S'ha escollit **Unity 3D** com al motor per dissenyar i desenvolupar el videojoc. Tot hi haver treballat amb el software prèviament, el motiu de la elecció es basa amb la disponibilitat de la documentació i les aportacions de la comunitat. Aquests dos elements han sigut imprescindibles per dur a terme el videojoc ja que sempre he trobat resposta a les preguntes que se'm plantejaven.

4.2. Unity 3D

Unity 3D és un *game engine* multiplataforma creat per Unity Technologies. L'empresa va ser fundada l'any 1988 pels tecnòlegs David Helgason, Nicholas Francis i Joachim Ante a Copenhague. L'èxit d'Unity es va donar quan van enfocar el seu motor a les necessitats dels desenvolupadors de videojocs independents. Van adoptar la política de "democratitzar el desenvolupament de videojocs", és a dir, volien que el seu contingut fos el més accessible possible per a tot tipus de creadors.

Els ***GameObjects*** són els objectes fundamentals d'Unity que representen personatges, accessoris i escenaris, és a dir, tots els objectes que formen el joc són *GameObjects*. Consta d'un objecte buit que funcionen com a contenidors de components, que són qui implementen les funcions i el comportament del objecte.

Els objectes del joc estan continguts dins una escena (***Scene***), i pot ser utilitzada per crear un menú principal, nivells individuals o qualsevol altra cosa. Cada escena constitueix una part del videojoc, on el programador incorporarà l'ambient, obstacles, decoració i el disseny dels components que aniran conformant el projecte.

El comportament d'un *GameObject* dins l'escena es controlat pels components que se li adjunten. Per implementar característiques pròpies, Unity permet crear un component propi anomenat ***Script.*** Aquest component dona la possibilitat de configurar i modificar propietats utilitzant els llenguatges de programació natius de l'editor.

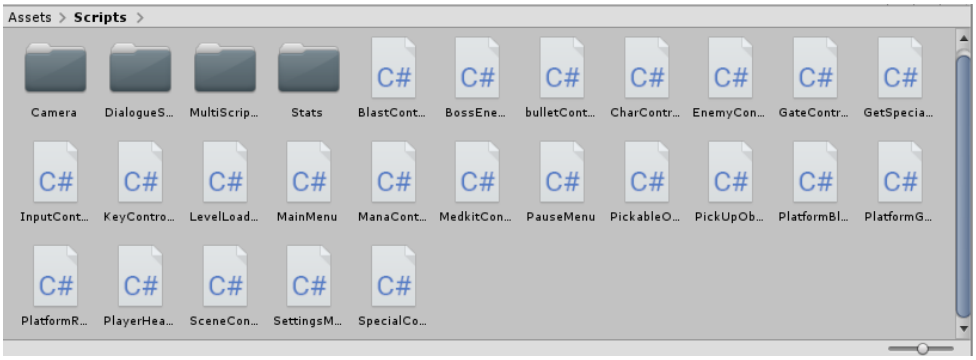


Figura 4.2 *- Carpeta Scripts de Unity 3D.*

La interfície d'Unity és fàcil de manipular i molt intuïtiva. L'editor permet organitzar les pestanyes de la interfície de manera lliure o bé utilitzant plantilles pròpies del software que venen per defecte amb la pestanya "Layout". Les pestanyes que formen la interfície d'Unity 3D són:

- **Hierarchy:** mostra la llista dels elements que es troben a l'escena del joc. És una estructura que permet l'accés als *GameObjects* afegits a l'escenari formant una jerarquia d'objectes.

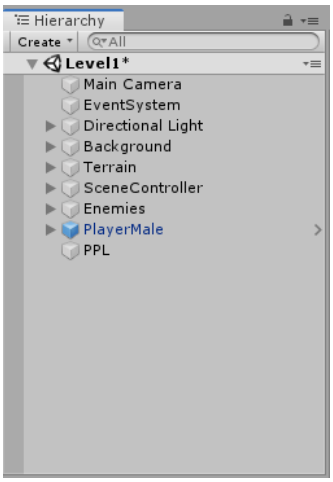


Figura 4.3 *- Hierarchy de la interfície de Unity 3D*

- **Inspector:** mostra informació detallada sobre el *GameObject* seleccionat. Permet visualitzar els components i les propietats de l'objecte, i modificar la seva funcionalitat a l'escena. S'utilitza per editar les propietats i la configuració de qualsevol element disponible a l'editor.

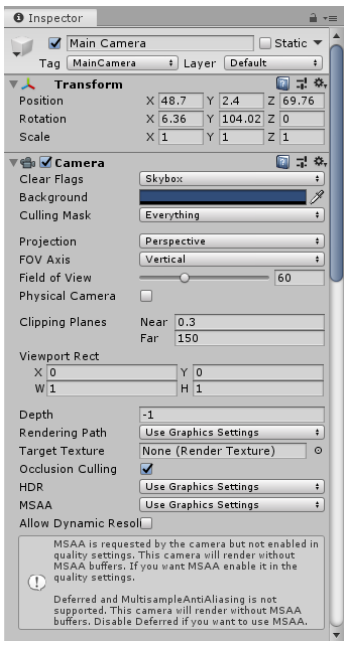


Figura 4.4 *- Inspector de la interfície de Unity 3D*

- **Project:** mostra l'estructura de carpetes del projecte com una llista de jerarquia. Quan es selecciona una de les carpetes, es mostra el contingut a la part dreta de la finestra. A les carpetes s'emmagatzemen els recursos que s'utilitzaran per dissenyar el videojoc.

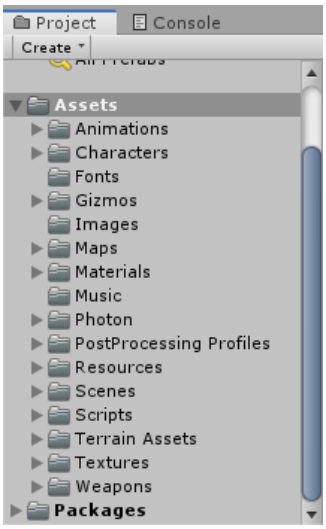


Figura 4.5 *- Project de la interfície de Unity 3D*

4.3. Altres Softwares

Per confeccionar alguns elements del videojoc i de la memòria, s'ha fet ús de altres softwares gratuïts com a eines per facilitar-ne el seu disseny. S'han utilitzats dos softwares per dissenyar els personatges principals. També s'han utilitzat editors per elaborar diagrames recollits en aquesta memòria.

- **Fuse:** software que permet crear i personalitzar personatges únics amb facilitat utilitzant una biblioteca amb contingut 3D de gran qualitat.
- **Mixamo:** ofereix suites de creació d'avatar, milers d'animacions per a escollir i un servei d'emparellament automàtic amb Fuse.
- **SmartDraw:** és una eina de dibuix estandarditzat que permet crear diagrames de tot tipus. Amb aquest software s'han dissenyat els diagrames UML i altres elements.
- **Creatly:** el software permet crear diagrames de flux i mapes mentals de forma senzilla i ràpida. Aquesta eina s'ha utilitzat per dissenyar el MockUp del projecte.

5. Anàlisi del videojoc

El projecte es va iniciar amb el disseny dels personatges principals, elaborant un primer esbós dels possibles models que podrien adoptar e i investigant plataformes per convertir-ho a un model 3D. Amb el prototip final incorporat a Unity3D, es va iniciar un procés de recerca per obtenir les animacions que farien possible el moviment i les accions de l'avatar. A mesura que avançava el projecte, es van anar afegint noves animacions que feien el personatge més complet millorant així la jugabilitat. També es va implementar un sistema de control per manipular les animacions de la màquina d'estats del jugador i poder-lo controlar dins un escenari de proves.

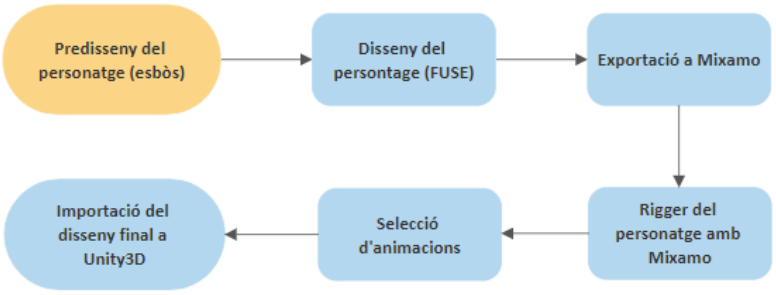


Figura 5.1 *- Diagrama de blocs del disseny dels personatges principals*

Per donar forma al videojoc i acotar les possibilitats es va iniciar el procés de creació de l'escenari principal. Es va fer un esbós de l'escena que identificava les zones amb accidents naturals i vegetació, una primera distribució dels enemics i la localització d'objectes rellevants per al transcurs de la història. El procés d'implementació de l'esbós a Unity3D es va dur a terme amb l'eina *Terrain*, un editor propi del *software* que permet crear escenaris amb multitud de paràmetres personalitzables. L'escena principal ha estat en constant evolució a mesura que s'incorporaven nous elements i es progressava amb el projecte.

Seguidament es va treballar amb els enemics del personatge principal. En aquest cas, el procés de disseny va ser diferent ja que els models 3D formen part d'un *asset* del *Asset Store*, però el sistema per incorporar les animacions és el mateix que amb el personatge principal. El repte que tenien aquests *GameObjects* era la implementació d'un sistema de control autònom que interaccionés amb el jugador quan es donin una sèrie de condicions. El sistema requeria que l'enemic es pugues moure lliurement per l'escenari evitant obstacles per arribar fins una destinació concreta. Posteriorment, amb aquestes característiques assolides, es van dissenyar un conjunt d'enemics amb una jerarquia per ser distribuïts arreu de l'escenari.

Quan es van completar els primers dissenys del personatge principal, els enemics i l'escenari, era necessària la creació d'elements que relacionen els tres blocs. Els objectes són la finalitat del jugador per avançar i els enemics tindran el paper de protegir-los intercedint amb el camí del jugador. El disseny dels objectes va ser progressiu a mesura que s'incorporaven noves funcionalitats al personatge principal i donant lloc a més possibilitats.

5.1. Personatges principals

Els personatges principals són els models que el jugador podrà escollir per ser controlats i navegar pel món virtual. S'explicarà el procés de disseny dels personatges, la elaboració de la màquina d'estats, el sistema de control que s'utilitzarà per manipular el personatge, així com els elements que conformen el conjunt.

5.1.1. Disseny

Per al disseny i construcció dels personatges principals, s'ha optat per utilitzar el software FUSE. Aquesta eina permet caracteritzar el teu model propi a partir de models 3D estàndards que ofereix el software. Et permet modificar multitud de paràmetres que formen les diferents parts del model: cap, ulls, galtes, orelles, coll, espatlla, pit, cames, peus... així com el disseny de vestuari del personatge.

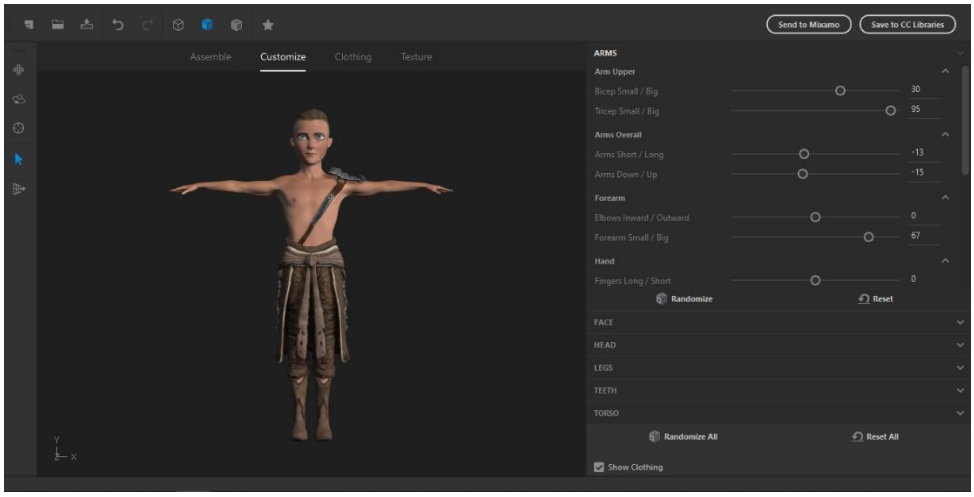


Figura 5.2 *- Disseny del model del personatge masculí amb Fuse*

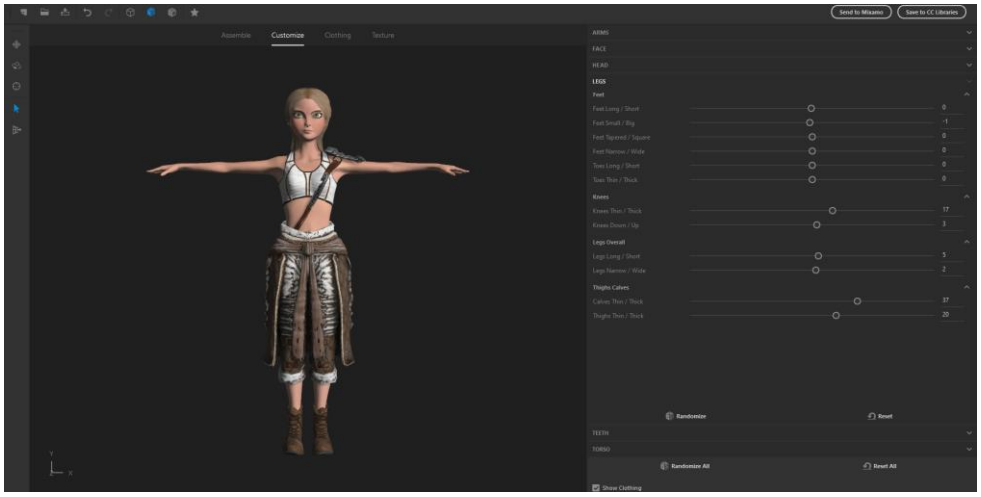


Figura 5.3 *- Disseny del model del personatge femení amb Fuse*

El motiu principal pel qual s'ha escollit FUSE com a editor dels personatges és el lligam que manté amb l'aplicació web MIXAMO. Aquesta *app* és capaç d'importar un model 3D provinent de FUSE (amb l'extensió *.obj*) amb l'objectiu de dotar-lo d'un sistema de control digital i poder incorporar animacions. El control digital és estrictament necessari per a executar animacions eficients al model. Tracta d'un sistema ossi que permet a les animacions identificar les articulacions amb les quals el personatge es dotarà de moviment anomenat *rigger*.

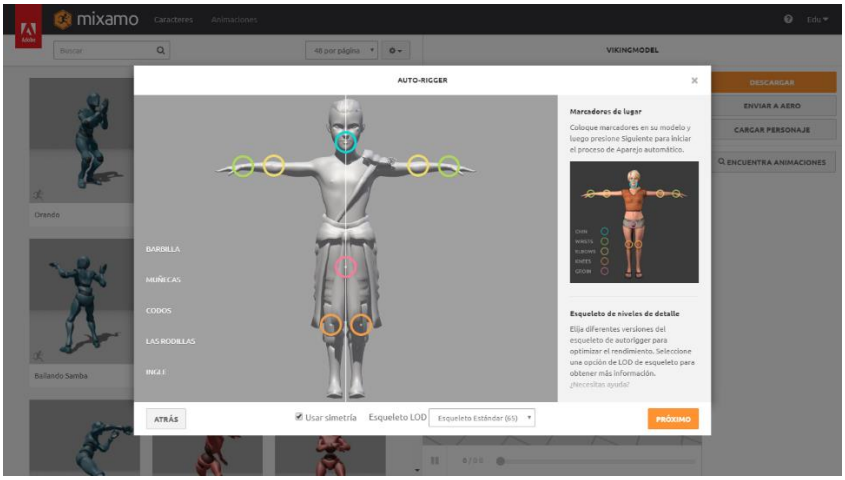


Figura 5.4 *- Auto-rigger del personatge amb Mixamo*

MIXAMO ofereix un sistema de *rigger* molt senzill d'implementar. Es basa principalment de cinc zones d'acció: barbeta, canells, colzes, engonal i genolls. Un cop s'identifiquen aquestes zones en el model, el software et construeix el sistema ossi automàticament. De vegades, el procés no es realitza correctament degut a la errònia situació de les zones d'acció, cal ajustar-les de nou i repetir el procés.

L'exportació del model a Unity3D (amb extensió *.fbx*) es recomana fer-ho amb T-Pose o posició en forma de T per evitar interferències quan s'incorporen noves animacions.

El sistema d'animació d'Unity té característiques especials per treballar amb models humanodies. És important configurar l'animació del tipus humanoide degut a que Unity proporciona un flux de treball personalitzat i un conjunt d'eines per a aquest tipus d'animacions. L'avatar de l'animació ve configurat per defecte amb la importació del model de Mixamo a Unity.

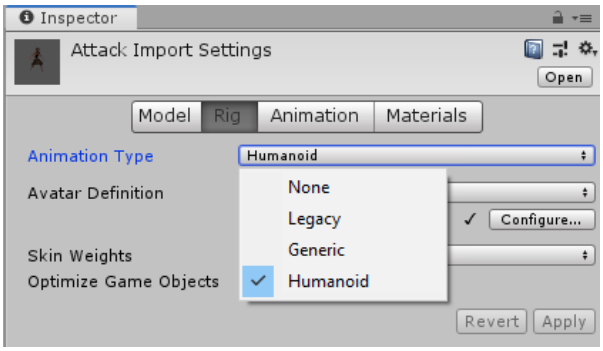


Figura 5.5 *- Configuració pestanya Rig d'una animació.*

A la pestanya *Animation* dins la configuració dels paràmetres d'una animació, s'han d'ajustar els paràmetres de *Root Transform* de la següent manera per evitar que la naturalesa de l'animació s'oposi al sistema de control. L'opció *Loop Time* es selecciona quan l'animació es repeteix infinitament mentre la condició per activar-la sigui cert. Les animacions de caminar o ajupir-se en són uns exemples.

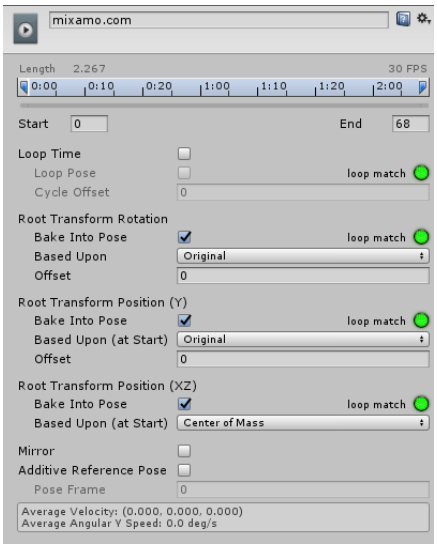


Figura 5.6 *- Configuració pestanya Animation d'una animació.*

Totes les animacions que s'importin de Mixamo s'han de configurar de la mateixa manera per a que el moviment es desenvolupi correctament

5.1.2. Màquina d'estats

Des de la app web MIXAMO s'obtenen els clips d'animació que permetran dissenyar la màquina d'estats del jugador. La màquina d'estats són un conjunt esdeveniments que composen les diferents accions que pot realitzar el personatge en funció de les condicions o restriccions de les transicions entre esdeveniments.

Cada estat representa una funció del jugador, en aquest cas, tindrem un estat per caminar, ajupir-se, saltar, posició d'atac, atacar, atac especial, bloquejar, interactuar, guanyar i morir, juntament amb l'estat de repòs (Idle) contingut dins l'estat "Walk". Aquesta funció s'executa a partir d'animacions que tenen un moviment associat per desenvolupar una acció concreta. Per canviar d'un estat a un altre es necessari tenir una condició, que s'implementa mitjançant transicions per tal de definir les seqüències possibles de les accions del personatge.

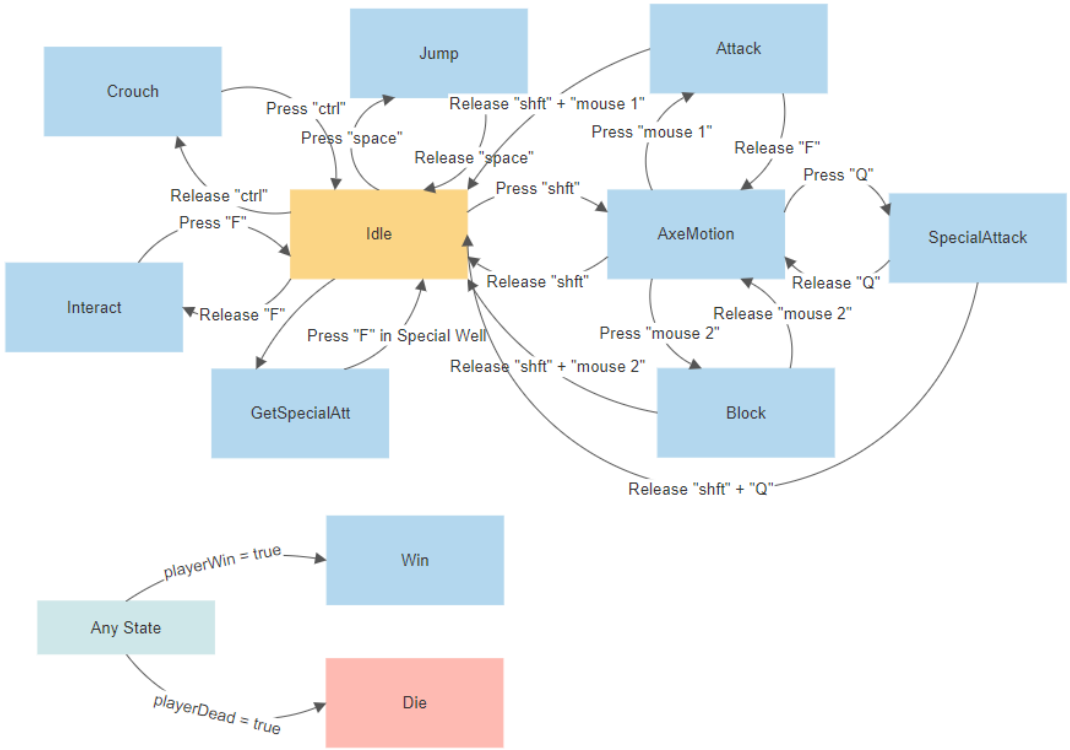


Figura 5.7 *- Màquina d'estats del jugador*

Descripció dels estats:

- **Walk:** estat d'entrada quan la unitat es troba en repòs. Aquest estat també conté les animacions per caminar amb totes direccions.
- **Jump:** quan el jugador salta.
- **Crouch:** estat actiu quan el jugador s'ajup i es mou en qualsevol direcció amb aquesta condició.
- **AxeMotion:** aquest estat indica que el jugador està amb la posició d'atac activa.
- **Attack:** realitzant el atac principal.
- **SpecialAttack:** llançant el atac especial.
- **Block:** l'estat que s'activa quan la unitat bloca els atacs dels enemics.
- **GetSpecialAtt:** estat que conté l'animació per aconseguir l'atac especial.
- **Interact:** s'activa quan el jugador interacciona amb un objecte.
- **Win:** aquest estat mostra amb una animació com a símbol de victòria.
- **Die:** estat actiu quan el jugador redueix la salut a cero.

L'estat inicial és el que entrarà per defecte el jugador al ser instanciat, en aquest cas, l'estat "Walk" és d'on deriven la resta de accions possibles del personatge. De l'estat per defecte, s'accedeix a les accions de saltar, ajupir-se, posició d'atac, aconseguir l'atac especial i interactuar. La condició per canviar d'estat es gestionada per la transició, que conté una variable booleana o un *trigger* que decideix si pot canviar a l'estat que es requereix o no. Un booleà et permet mantenir el valor lògic, en canvi, un *trigger* funciona com un disparador que només s'executa un cop. El conjunt de variables que decideixen quina transició activar, entre d'altres, formen una llista de paràmetres que són controlats mitjançant *Scripts*.

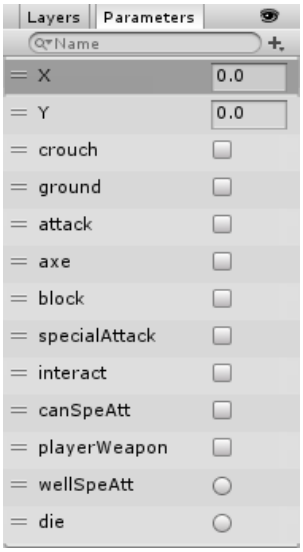


***Figura 5.8** - Llista de paràmetres del jugador*

Per donar prioritat a una acció es pot accedir a un estat a partir de *Any State* on, sigui quin sigui l'estat actiu, canviarà si es dóna la condició necessària.

Cada estat conté una animació o conjunt d'animacions que representen una acció. L'estat per defecte conté una mescla d'animacions interconnectades entre elles mitjançant un arbre de mescla, més conegut com *Blend Tree*. Aquesta eina permet mesclar dos o més moviments per crear una animació suau, sempre i quan els moviments a mesclar siguin d'una naturalesa similar i de temps. Els estats *"*Crouch*"* i *"*AxeMotion*"* també els formen un arbre de mescla.

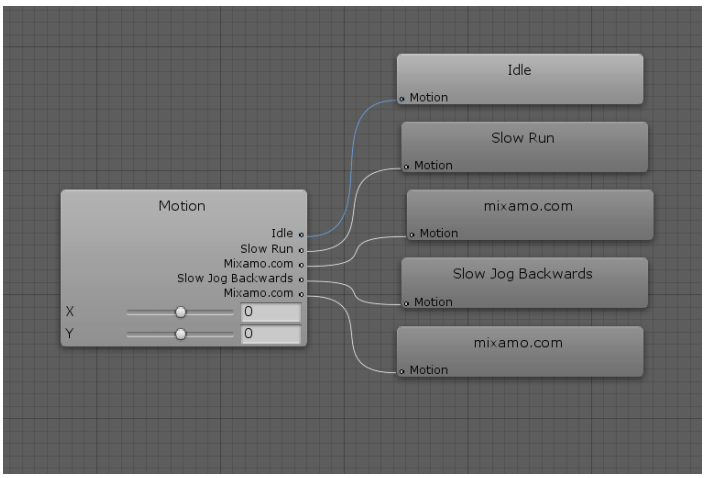


Figura 5.9 *- Arbre de mescla complet del enemic*

L'estat "Walk" conté cinc animacions que es distribueixen en un plànol 2D acotat entre 1 i -1. Cada animació s'associa a un punt del plànol representant la direcció del moviment del jugador. L'animació "Idle" correspon a la posició (0,0) degut a que es l'estat de repòs. La resta d'animacions s'enllacen amb les possibles direccions que el personatge pot caminar. Els estats "Crouch" i "AxeMotion" el formen un *Blend Tree* amb les mateixes característiques, una animació "Idle" juntament a les quatre animacions que completen el moviment del jugador.

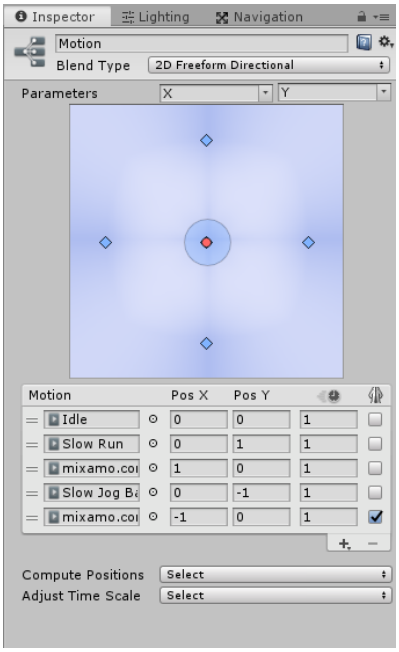


Figura 5.10 *- Configuració dels inputs seleccionats*

5.1.3. Controls bàsics

El sistema de control escollit es basa amb el mètode input. Unity 3D ofereix la possibilitat de configurar una sèrie d'entrades identificades amb un nom, el botó a prémer per tal d'activar l'input i una sèrie de paràmetres relacionats amb la sensibilitat d'activació, entre d'altres. Cada una de les entrades es configuren en funció del tipus d'acció a realitzar, és a dir, hi ha accions que requereixen mantenir premut el botó per desenvolupar correctament l'animació o d'altres que tan sols és necessari prémer un cop. Quan es prem el botó assignat, activa una variable booleana que permet activar la transició de la màquina d'estats i canviar d'un estat a un altre. Per tant, es configuren quatre tipus d'inputs:

- *Down*: s'activa mantenint premut el botó.
- *Up*: s'activa quan es deixa de mantenint premut el botó.
- *Pressed*: s'activa quan es prem un cop el botó.
- *Release*: s'activa quan es deixa de prémer un cop el botó.

Figura 5.11 *- Configuració dels inputs seleccionats*

S'ha optat per configurar les entrades que controlaran el moviment i les accions del personatge principal de la següent manera.

Es mourà el model amb les tecles W (moviment cap a avant), S (moviment cap a enrere), D (moviment cap a la dreta) i A (moviment cap a l'esquerra). S'utilitzarà la tecla *Control* per a que el personatge s'ajupi i *Space* per a saltar.

Pel que fa al sistema d'atac, s'ha implementat una posició especial de guàrdia per incrementar la precisió dels atacs disminuint la velocitat del personatge mentre mantens premut la tecla *Shift*. En aquesta condició, s'ha assignat el botó esquerre del ratolí per realitzar l'atac principal i el dret per crear un escut que et protegeix dels atacs dels enemics. També s'ha configurat la tecla Q per llençar l'atac especial, sempre i quan el jugador estigui amb la posició d'atac activa.

S'ha assignat la tecla F per interactuar amb objectes i estructures de l'escena. Es permetrà interaccionar amb els objectes quan aquests tinguin les condicions necessàries per poder ser manipulats.

L'*script* que identifica les tecles que prem l'usuari i les enllaça amb l'acció configurada al *script* principal *CharController.*

Per concentrar i modificar amb facilitat les característiques del personatge principal, s'ha creat un *script* anomenat *CharStats*. Des de l'*script CharController* es crida a la classe *CharStats* per accedir als atributs del jugador.

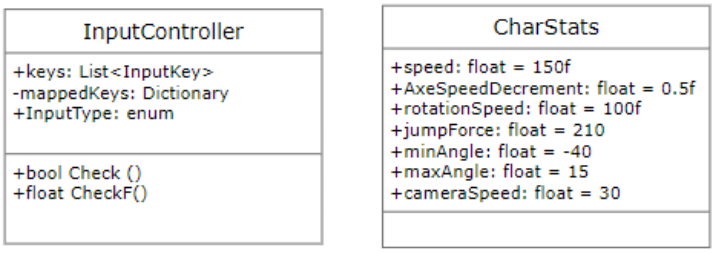


Figura 5.12 *- UML dels scripts InputController i CharStats*

5.1.4. Control de la càmera

Per implementar la càmera en tercera persona al *software Unity3D,* s'ha fet ús de dos *GameObjects* que defineixen la posició, l'altura i rotació de la càmera respecte el personatge principal anomenats *CameraShoulder* i *CameraHolder*. Amb *l'script CharController* s'ha configurat la *MainCamera* del projecte segons la disposició de les referències.

```
private void CameraControl()
{
    rotY += mouseDelta.y * deltaT * Stats.rotationSpeed;
    float xrot = mouseDelta.x * deltaT * Stats.rotationSpeed;

    tr.Rotate(0, xrot, 0);

    rotY = Mathf.Clamp(rotY, Stats.minAngle, Stats.maxAngle);

    Quaternion localRotation = Quaternion.Euler(-rotY, 0, 0);
    CameraShoulder.localRotation = localRotation;

    cam.position = Vector3.Lerp(cam.position, CameraHolder.position, Stats.cameraSpeed * deltaT);
    cam.rotation = Quaternion.Lerp(cam.rotation, CameraHolder.rotation, Stats.cameraSpeed * deltaT);
}
```

Figura 5.13 *- Mètode CameraControl() del script CharController.*

Es defineixen dues variables flotants (*xrot, rotY*) que estableixen la velocitat de rotació de la càmera i acoten l'amplitud d'aquesta. Es configura l'angle de rotació de la càmera amb el comandament *Quaternion.Euler* i assignant aquest valor al *GameObject CameraShoulder.* Finalment s'interpola la posició i la rotació de la càmera amb les coordenades de consigna emmagatzemades al *GameObject CameraHolder.* La complexitat d'aquest sistema es deu als requeriments de control del jugador, basat amb la rotació de la càmera segons el moviment del ratolí.

5.1.5. Components

Els components que s'han utilitzat per donar física als models 3D i controlen les accions que desenvolupen són:

- **Transform:** aquest component determina la posició, rotació i escala d'un objecte a l'escena. Transforma la posició en les coordenades X,Y,Z mesurada amb *world unit*; la rotació al voltant dels eixos X,Y,Z mesurada amb graus i l'escala al llarg dels eixos X,Y,Z sent el valor 1 la mida original del model.

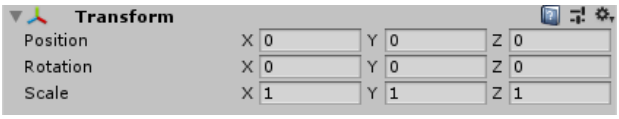


Figura 5.14 *- Component Transform d'un GameObject*

- **Rigidbody:** el component s'encarrega d'habilitar que un cos rígid actuï baix els efectes de la física implementada al software.

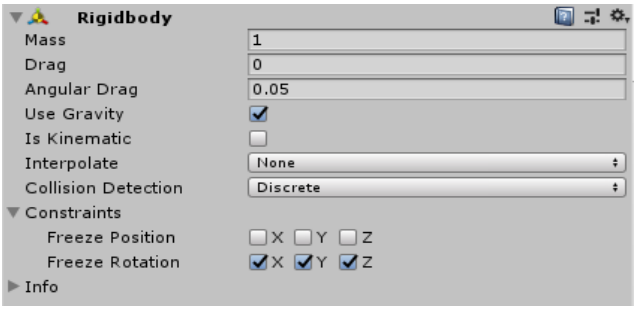


Figura 5.15 *- Component Rigidbody d'un GameObject*

El motor de física que utilitza Unity 3D és el NVIDIA PhysX, i és imprescindible implementar-lo per a que qualsevol *GameObject* estigui influenciat per la gravetat i per altres forces addicionals provocades per la interacció amb altres objectes que contenen també un cos rígid. Quan es vol aplicar física a un cos rígid de dimensions reduïdes que es mou a grans velocitats s'utilitza l'opció *Is Kinematic,* per exemple un projectil.

Cal destacar que quan s'utilitza el component *Rigidbody* és important que els càlculs de física es realitzin en cada instant independentment de la velocitat de fotogrames del videojoc. Per aquest motiu, es substitueix la funció *Update()* per *FixedUpdate().*

- **CapsuleCollider:** Existeixen tres tipus de col·lisionadors en funció de la forma que adopten: *box collider, sphere collider, capsule collider i mesh collider*. Es decideix el tipus segons la naturalesa del model, en aquest cas, s'utilitzarà *capsule collider*. Aquesta varietat està formada per dos mitges esferes a cada extrem del model i és l'estructura que més s'adapta a la forma d'un humanoide.

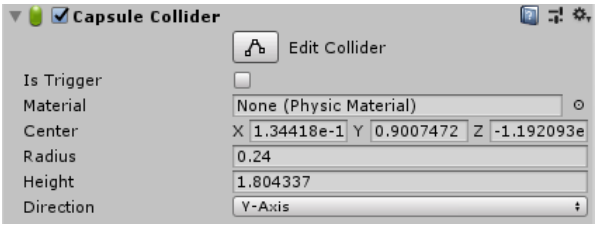


Figura 5.16 *- Component Capsule Collider d'un GameObject*

El component *Collider* s'afegeix a un objecte per determinar els límits d'aquest i així evitar que s'envaeixi l'espai d'un altre objecte. També s'utilitza per comprovar quan dos objectes col·lisionen entre ells.

La pestanya *Is Trigger* es marca quan un cos rígid actuï com un disparador i no enregistri la col·lisió, sinó que detecti una interferència entre dos cossos rígids. Es determina l'àrea d'efecte amb els paràmetres *center* i *radius,* i es contempla la intersecció mitjançant les funcions *OnTriggerEnter(), OnTriggerExit() y OnTriggerStay().*

- **Animator:** és l'encarregat de controlar el sistema d'animacions d'un model. Es requereix d'un controlador que contingui el conjunt d'animacions que formen el moviment del objecte, juntament amb les transicions entre estats. També es necessari enllaçar l'avatar de l'objecte que executarà les animacions del controlador assignat a l'Animator.

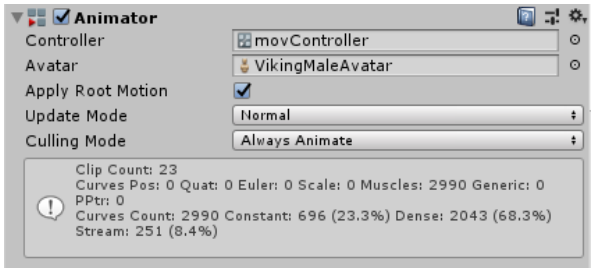


Figura 5.17 *- Component Animator d'un GameObject*

- **AudioSource**: aquest component s'encarrega de reproduir sons en un entorn 3D. Es imprescindible afegir el component AudioListener en algun objecte de l'escena per poder reproduir els sons del AudioSource.

- **PhotonAnimatorView:** component necessari per sincronitzar les animacions del mode multijugador amb Photon.

- **PhotonTransfromView:** component que sincronitza la posició, rotació i escala del *GameoObject* al mode multijugador amb Photon.

- **PhotonView:** aquest component es necessari per a la connexió multijugador amb Photon.

(El capítol nº8 de la present memòria explica amb més detalls els components que deriven de Photon Network)

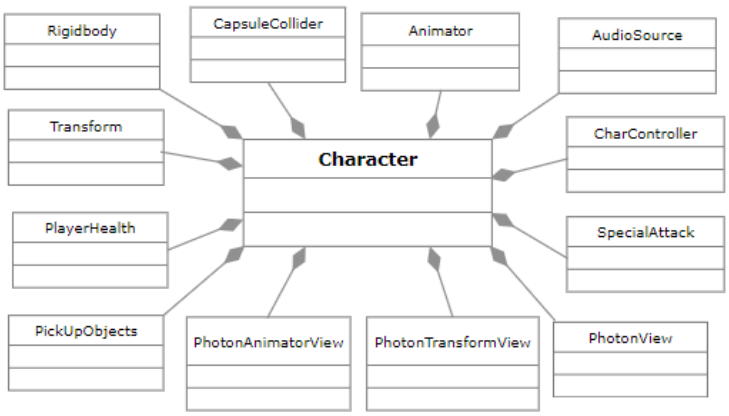


Figura 5.18 *- Components del personatge principal.*

5.1.6. Scripts

Els *Scripts* que s'han implementat al personatge principal són:

- **CharController:** és l'encarregat de gestionar el sistema d'inputs que governen el moviment del avatar mitjançant variables booleanes. S'encarrega també de determinar la posició de la càmera i de processar les característiques del personatge.

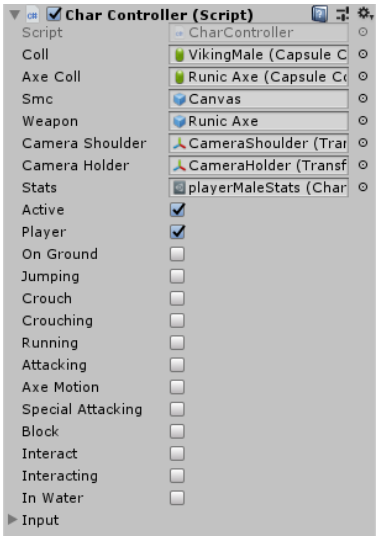


Figura 5.19 *- Component Script CharController del personatge principal.*

Descripció de les variables

- **coll:** component *CapsuleCollider* del personatge principal.
- **axeColl:** component *CapsuleCollider* de l'arma principal.
- **smc:** *Canvas* on hi ha dibuixat el punt de mira.
- **weapon:** arma principal del jugador.
- **CameraShoulder:** posició de la càmera respecte l'espatlla del jugador.
- **CameraHolder:** posició de la càmera respecte CameraShoulder.
- **cam:** es referència la *Main Camera* de l'escena.
- **Stats:** referència de *l'script CharStats* per crida variables de la classe.
- **Player:** booleà que confirma que el personatge es controlat per l'usuari.
- **OnGround:** indica si el jugador està tocant el terra de l'escenari.
- **Jumping:** indica si el jugador està saltant.
- **Crouch:** indica si el jugador està ajupit.
- **Attacking:** indica si el jugador esta atacant.
- **AxeMotion:** indica si el jugador es troba amb posició d'atac.

- **Block:** indica si el jugador es troba amb posició de bloqueig.
- **Interact:** indica si el jugador està interactuant amb un objecte.

Descripció dels mètodes principals

- **PlayerControl():** mètode que executa el sistema d'*inputs* que permeten el control del moviment del jugador.
- **CameraControl():** estableix la posició, la velocitat de rotació i els límit de la càmera respecte el jugador.
- **MoveControl():** en aquest mètode es determina la direcció i les restriccions del moviment del jugador en funció de l'estat actiu. També s'executa l'activació de variables controlades per booleans.
- **AnimControl():** és l'encarregat d'executar les animacions del jugador.

A l'annex A3 (Codis font) es mostra el contingut de *l'script CharController*.

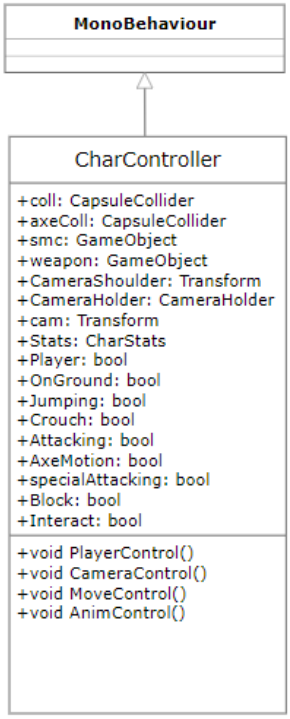


***Figura 5.20** - UML dels Scripts CharController.*

- **PlayerHealth:** recull el conjunt de característiques que defineixen la salut del jugador. També gestiona els efectes visuals quan el jugador rep un atac o bé recupera vida. Mitjançant la variable booleana *playerDead* es gestiona la mort del jugador.

Figura 5.21 *- Component Script PlayerHealth del personatge principal.*

Descripció de les variables

- **currentHealth:** variable que emmagatzema la salut actual del jugador.
- **startingHealth:** correspon a la salut inicial del jugador.
- **healthSlider:** conté la barra de vida del HUD del jugador.
- **damage/healingImage:** emmagatzemen les imatges que formen part de l'estètica del HUD.
- **healing:** indica quan el jugador està recuperat salut.
- **damaged:** indica quan el jugador està rebent l'atac d'un enemic.
- **playerDead:** booleà que indica si el jugador ha mort o no.

Descripció dels mètodes principals

- **TakeDamage():** mètode que es crida quan es resta salut al jugador.
- **GetMedkit():** mètode que incrementa la salut del jugador.
- **Death():** mètode que es crida quan el jugador perd la vida.

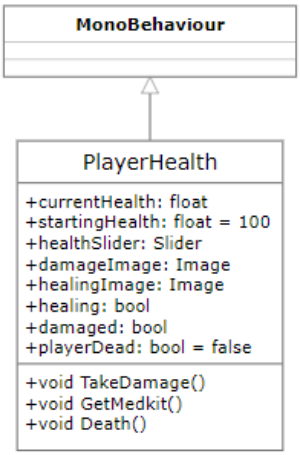


Figura 5.22 *- UML del script PlayerHealth*

- **PickUpObjects:** aquest Script recull el conjunt d'icones que es mostren a la UI del jugador i les gestiona segons les condicions que s'imposen. Es l'encarregat d'identificar els objectes que es troben al davant i d'afegir-lo a l'inventari del personatge. També gestiona un conjunt de variables booleanes que decideixen el transcurs de la història.

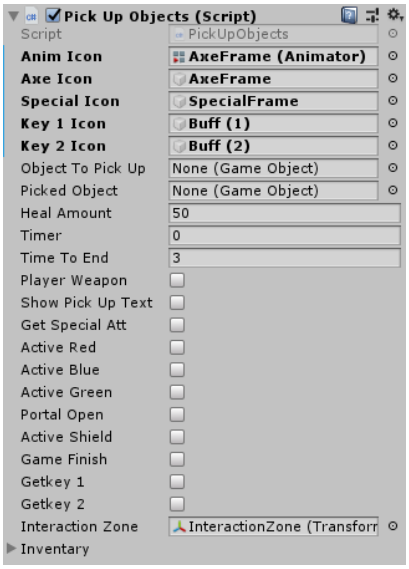


Figura 5.23 *- Component Script PickUpObjects del personatge principal.*

Descripció de les variables

- **animIcon:** referència l'*Animator* de les icones del HUD.
- **axeIcon, specialIcon, key1Icon, key2Icon:** emmagatzemen les icones del HUD.

- **ObjectToPickUp:** és la referència de l'objecte que té al davant el jugador intercedint amb el *collider InteractionZone.*
- **PickedObject:** indica quin és l'últim objecte que ha interaccionat el jugador
- **healAmount:** quantitat de salut que es restaura al jugador.
- **playerWeapon:** indica si el jugador té l'arma principal al seu inventari.
- **showPickUpText:** indica si es permet mostra el text d'interacció.
- **getSpecialAtt:** indica si el jugador ha aconseguit l'atac especial.
- **activeRed, activeBlue, activeGreen:** indica si s'ha interaccionat amb els cubs del enigma.
- **portalOpen:** indica si el portal s'ha desbloquejat.
- **activeShield:** indica si s'ha d'activar el escut protector.
- **gameFinish:** indica si s'ha acabat el joc.
- **getKey1, getKey2:** indica si el jugador conté les claus al seu inventari.
- **interactionZone:** referencia la zona d'interacció que permet identificar els objectes al jugador.
- **inventary:** emmagatzema una llista dels objectes obtinguts pel jugador.

Descripció dels mètodes principals

- **Update():** al mètode Update s'executa la identificació d'objectes i la presa de decisions en funció del objecte que es té al davant.

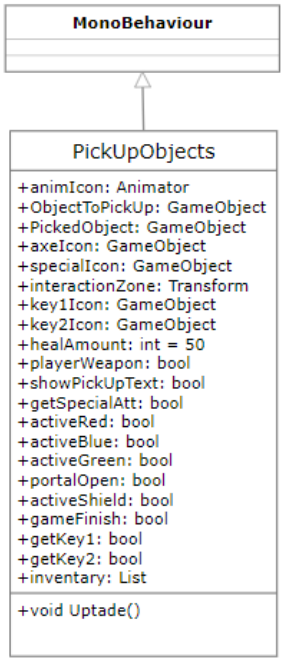


Figura 5.24 *- UML del script PickUpObjects*

- **SpecialController:** s'implementa al jugador per poder controlar l'atac especial. Se li assigna la posició de sortida del projectil, el tipus de projectil i l'explosió que instanciarà quan aquesta xoca contra un objecte rígid.

Figura 5.25 *- Script SpecialController del personatge principal.*

Descripció de les variables

- **shootPoint:** referencia la posició de sortida de l'atac especial.
- **bulletPrefab:** emmagatzema el *prefab* que conté l'atac especial
- **explosionVFX:** emmagatzema el *prefab* que conté l'efecte especial quan el jugador aconsegueix l'atac especial.

Descripció dels mètodes principals

- **Shoot():** mètode que instància l'atac especial del jugador quan les condicions ho permetin.
- **GetSpecialAttack():** mètode que instància l'efecte especial quan el jugador interacciona amb l'objecte que li dona la capacitat de realitzar un atac especial.

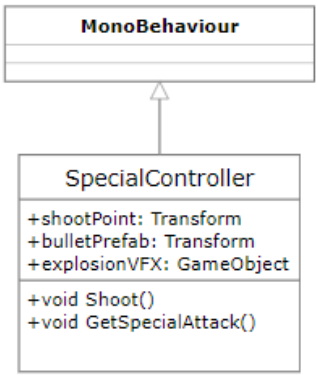


Figura 5.26 *- UML del script SpecialController.*

5.2. Enemics

En un videojoc, els enemics són aquells personatges que tenen com a objectiu oposar-se al camí del jugador per dificultar el seu recorregut. Tots els enemics comparteixen les mateixes consignes, representen un conflicte amb el personatge i contribueixen en la immersió de la historia.

5.2.1. Disseny

S'han escollit uns dissenys d'ús lliure provinents de l'*Asset Store* que ofereixen diferents estils d'enemics però amb una notable relació estètica. Segons les dimensions i la caracterització dels models 3D, s'ha dissenyat una jerarquia d'enemics classificats amb cinc nivells. El nivell de cada enemic varia en funció de les seves característiques pròpies d'atac i salut. S'ha implementat aquesta estructura per diferenciar el tipus d'adversari segons la fita a obtenir. A més, també afegeix credibilitat i dinamisme al videojoc ja que implica un augment de dificultat i una mostra de progrés.

	AttackDamage	**Health**	**TimeBetweenAttacks**
Nivell 1	10	10	2.5
Nivell 2	10	20	2.5
Nivell 3	15	25	2.5
Nivell 4	15	25	3
Nivell 5	10	40	3

Taula 5.1 *- Atributs dels enemics segons el nivell.*

El nivell 1 el formaran els enemics que més abunden al escenari. Són els peons dins la jerarquia d'enemics i per tant, són els enemics amb pitjors atributs respecte la resta de nivells.

Figura 5.27 *- Model 3D del enemic de nivell 1.*

El següent nivell el formen els enemics que acompanyen als de nivell 1. S'utilitzaran per emfatitzar zones d'interès a l'escena. Els enemics de nivell 2 tenen el doble de salut respecte els del nivell inferior.

Figura 5.28 *- Model 3D del enemic de nivell 2.*

El nivell 3 correspon als enemics que apareixen quan el jugador assoleix un objectiu principal. Aquest model té una gran envergadura per justificar l'augment de salut i l'increment de la potència d'atac.

Figura 5.29 *- Model 3D del enemic de nivell 3.*

Els enemics d'aquest nivell tenen una funció similar als del nivell anterior. Apareixeran quan el jugador assoleixi una fita més difícil de aconseguir que l'anterior. S'ha incrementat la freqüència d'atac dels enemics d'aquest nivell per augmentar la dificultat del progrés.

Figura 5.30 *- Model 3D del enemic de nivell 4.*

El nivell 5 correspon a disseny de l'enemic final. Es trobarà al final de la història i tindrà habilitats que no comparteix amb la resta d'enemics. L'enemic estarà estàtic en un lloc inaccessible per al jugador i se li implementarà l'habilitat de llençar un projectil que afecti a la salut del jugador. Per això, s'ha incrementat la freqüència d'atac i la salut de l'enemic però la potència d'atac no s'ha vist afectada.

Figura 5.31 *- Model 3D del enemic de nivell 5.*

5.2.2. Màquina d'estats

Per configurar les animacions del enemic s'ha dissenyat una màquina d'estats però en aquest cas, no s'ha fet ús dels arbres de mescla ja que les accions que ha de desenvolupar l'enemic són més simples.

Les transicions contenen condicions que depenen únicament de les accions del jugador. Les condicions per canviar d'un estat a un altre també són dirigides a partir de variables booleanes controlades mitjançant *Scripts*.

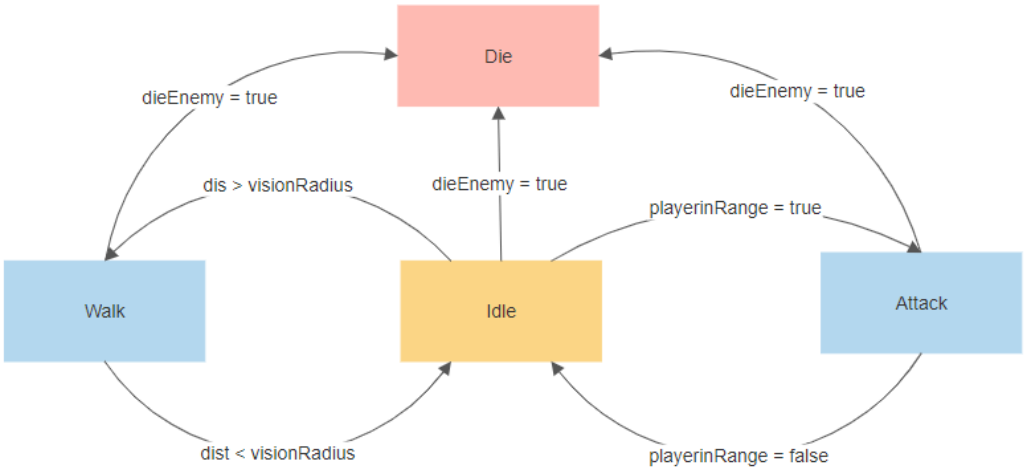


Figura 5.32 *- Màquina d'estats dels enemics.*

Descripció dels estats:

- **Idle:** estat d'entrada quan la unitat es troba en repòs. D'aquest estat se'n deriven la resta.
- **Walk:** estat que s'activa quan el jugador entra dins el radi d'acció de l'enemic. Mentre aquesta condició sigui certa, l'enemic caminarà cap al jugador. En cas de sortir de la zona d'acció, es retornarà a l'estat inicial Idle.
- **Attack:** s'activa quan el jugador està lo suficientment prop de l'enemic per poder-lo atacar.
- **Die:** estat actiu quan la salut de l'enemic és igual o inferior a zero.

L'enemic entrarà a l'estat "Idle" per defecte i d'allí en deriven tres accions més. En aquest cas, no s'ha fet ús de l'estat "Any State" ja que la simplicitat de l'estructura permet enllaçar fàcilment l'estat de morir "Die" amb la resta d'accions.

La llista de paràmetres està formada de tres booleans que activen de les transicions corresponents.

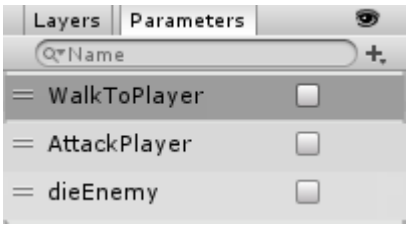


Figura 5.33 *- Llista de paràmetres del enemic*

5.2.3. Intel·ligència artificial

Per controlar el moviment de l'enemic s'ha optat per utilitzar el mètode Nav Mesh Agent. Aquesta eina permet establir un destí on el model que conté aquest component es dirigirà quan es donen les condicions establertes. Abans, és important conèixer el funcionament del NavMesh.

Per determinar la trajectòria possible que pot recórrer l'agent que conté el component, és imprescindible crear una malla que construeixi la geometria del nivell anomenada NavMesh. En primer lloc, cal determinar quins són els límits i obstacles del mapa així com la geometria de l'escena que pot afectar a la lliure navegació de l'agent. En segon lloc, es configuren les característiques del model en qüestió. S'ajusta el radi de seguretat, com de prop pot està l'agent d'un obstacle; l'altura, que defineix quina alçada pot superar; el pendent, defineix la màxima inclinació que pot recórrer; i l'altura dels obstacles, que defineix fins quina alçada pot abastar.

Un cop identificades les restriccions del escenari, es procedeix a la construcció de la malla, anomenat aquest procés com *Bake.* En aquest punt, ja tenim definit l'itinerari que pot recorrer l'agent en qüestió i es mostra amb un ombrejat de color blau.

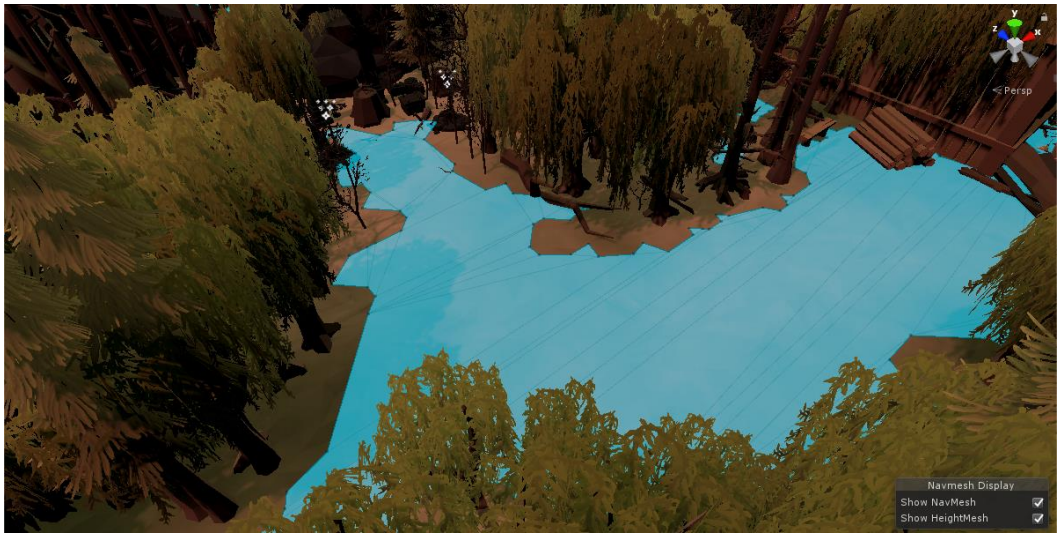


Figura 5.34 *- NavMesh del enemic*

L'objectiu principal d'establir el camí de l'enemic és perseguir al jugador per atacar-lo seguint la trajectòria correcta. NavMeshAgent permet establir un *target* com a destí del seu recorregut amb la funció *SetDestination()*. Aquesta funció es configura amb un *Script* i és l'acció que es durà a terme quan la variable booleana que activa la transició *Idle* → *Walk* sigui certa. Aquest serà el primer pas per iniciar el sistema d'atac que permetrà afectar la salut del jugador quan es donen les condicions corresponents.

5.2.3.1. Sistema d'atac

El sistema d'atac dissenyat per a l'enemic es basa en la detecció d'un cos rígid dins d'un radi d'acció. Es crea la funció *FollowPlayer()*, que consta d'un vector tridimensional que calcula la distància entre la posició del jugador i la pròpia del enemic. S'introdueix també una variable flotant que estableix la distància del radi de detecció. Quan la distància entre els dos personatges sigui menor al radi de detecció es canviarà el valor lògic del booleà que activa la transició que canvia a l'estat de caminar. Per determinar quin és camí per arribar fins l'objectiu (*target*) es crida a la funció *SetDestination().*

Quan el *trigger* del jugador entra en contacte amb el *CapsuleCollider* de l'enemic, es crida a la funció d'atacar. Mitjançant la funció *OnTriggerEnter()* es comprova si el cos rígid que intercepta el seu propi *Collider* és el jugador i si és així, les variables booleanes *playerInRange* i *attackPlayer* canviaran el valor lògic i activaran la funció i la transició d'atacar a la màquina d'estats. Quan l'enemic està lo suficientment prop com per a realitzar un atac, es crida a la funció *Attack()* que conté un temporitzador que regula els intervals de temps en que l'enemic pot ferir al jugador.

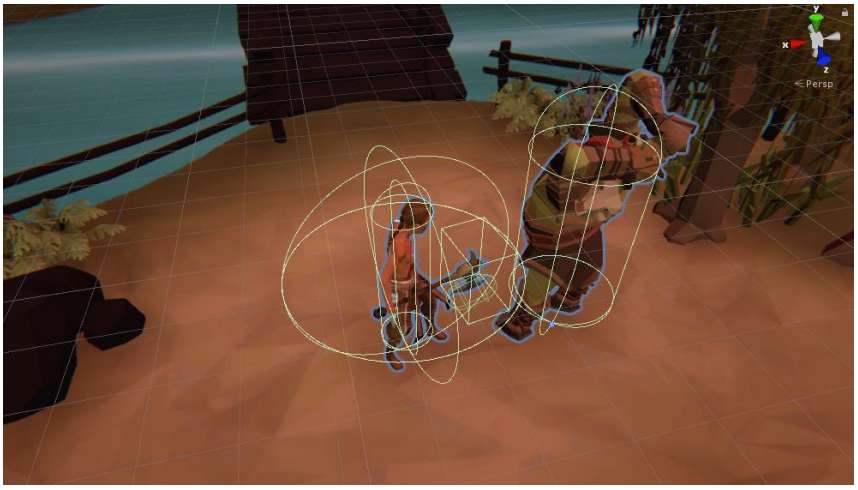


Figura 5.35 *- Intersecció dels colliders del jugador i enemic.*

Si el jugador surt de la intersecció entre ell i l'enemic, aquest deixarà d'atacar i canviarà a l'estat de caminar, per tal de buscar novament que el jugador estigui en rang d'atac. Quan el jugador surt del radi de detecció o bé la salut del jugador s'esgota, tornaran a canviar els valors lògics dels booleans corresponents activant la transició a l'estat idle.

5.2.4. Components

Els components que s'han utilitzat per donar física als models 3D i controlen les accions que desenvolupen són:

- **Transform:** determina la posició, rotació i escala d'un objecte a l'escena
- **Rigidbody:** el component s'encarrega d'habilitar que un cos rígid actuï baix els efectes de la física implementada al software.
- **CapsuleCollider:** el component *Collider* s'afegeix a un objecte per determinar els límits d'aquest i així evitar que s'envaeixi l'espai d'un altre objecte.
- **Animator:** és l'encarregat de controlar el sistema d'animacions d'un model.
- **AudioSource**: aquest component s'encarrega de reproduir sons en un entorn 3D.
- **Nav Mesh Agent:** aquest component s'ha implementat als enemics per dotar-los de moviment autònom per l'escena. Els models amb Nav Mesh, es mouran cap a l'objectiu evitant obstacles de l'escenari i altres agents que continguin aquest component que també busquin arribar a la destinació imposada. Aquesta eina de navegació utilitza *Pathfinding* per actualitzar constantment els camins possibles per arribar fins al seu *target.*

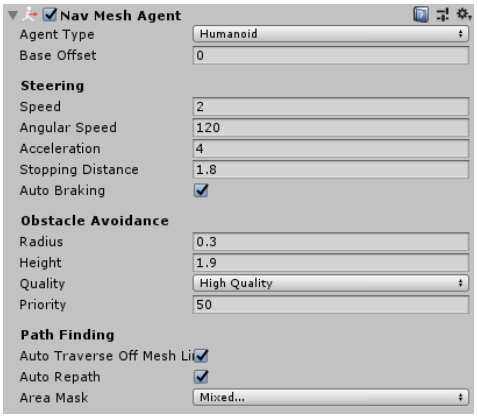


Figura 5.36 *- Component NavMesh Agent dels enemics.*

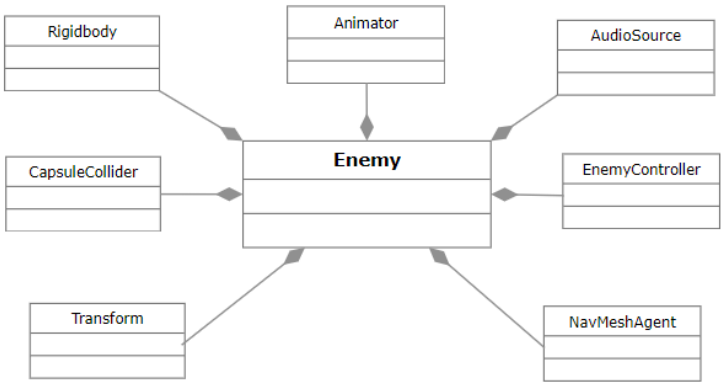


Figura 5.37 *- Components del enemic*

5.2.5. Scripts

S'han creat dos controladors per governar i gestionar les variables que defineixen el moviment i comportament dels enemics. La principal diferència entre els dos Scripts és la implementació del sistema d'atac que adopta l'enemic. S'utilitzarà un controlador per als enemics amb atac cos a cos i un altre per als enemics amb atac a distància.

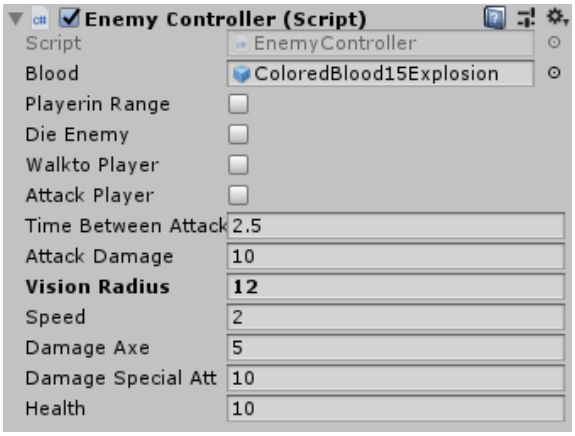


Figura 5.38 *- Scripts EnemyController i BossEnemyController dels enemics.*

Dins del conjunt d'enemics amb sistema d'atac cos a cos, segons la seva funció a l'escena, s'utilitzarà *l'Script EnemyController* i es modificaran les variables *Speed, Vision Radius i Attack Damage* amb l'objectiu de diferenciar la dificultat de l'enemic segons la velocitat de moviment, la distància de detecció i la intensitat dels seus atacs.

Descripció de les variables

- **blood:** emmagatzema el *prefab* que s'instancia quan el jugador fereix l'enemic.
- **playerInRange:** indica quan el jugador es troba a rang de l'enemic.
- **dieEnemy:** indica quan l'enemic mort.
- **walkToPlayer:** indica quan l'enemic pot avançar direcció al jugador.
- **attackPlayer:** indica quan l'enemic pot atacar al jugador.
- **timeBetweenAttacks:** variable que indica la freqüència d'atac del enemic.
- **attackDamage:** referencia la intensitat dels atacs dels enemics.
- **visionRadius:** marca la distància mínima de detecció per a que l'enemic avanci cap al jugador.
- **speed:** velocitat de moviment de l'enemic.
- **damageAxe:** quantifica la intensitat de l'atac principal del jugador.
- **damageSpecialAtt:** quantifica la intensitat de l'atac especial del jugador.
- **health:** indica la salut de l'enemic.

Descripció dels mètodes principals

- **OnTriggerEnter():** mètode que gestiona les accions quan el jugador entra dins el radi d'acció de l'enemic i s'encarrega també de reduir la salut de l'enemic quan es realitza un atac sobre ell.
- **OnTriggerExit():** mètode que canvia el valor lògic de les variables booleanes que decideixen quan es permet atacar al jugador.
- **FollowPlayer():** aquest mètode crida la funció *SetDestination()* quan la distància del jugador respecte l'enemic és més petita que la distància mínima de detecció.
- **Attack():** mètode que s'encarrega de restar salut al jugador segons la freqüència establerta.
- **SoundControl():** mètode que gestiona els sons de l'enemic.
- **AnimControl():** mètode que gestiona les animacions de l'enemic.
- **Die():** mètode que es crida quan la salut de l'enemic és igual o menor a zero.

A l'annex A3 (Codis font) es mostra el contingut de *l'script EnemyController*.

Pel que fa als enemics amb el sistema d'atac a distància, s'ha definit el projectil i la seva posició de sortida i s'ha afegit la variable booleana que decideix si l'enemic està en disposició d'atacar al jugador. També s'ha incrementat la vida d'aquest tipus d'enemics respecte el conjunt d'enemics amb sistema d'atac cos a cos.

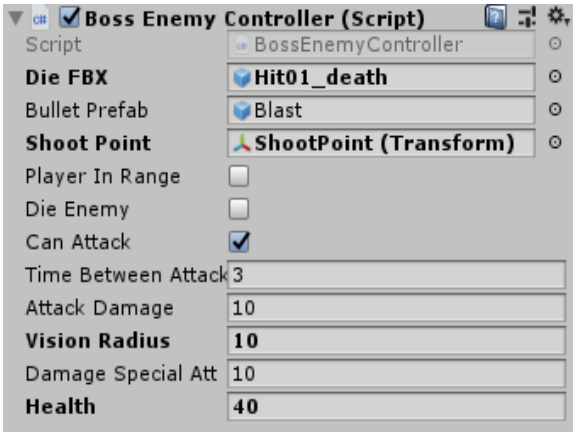


Figura 5.39 *- Script BossEnemyController del enemic final.*

Descripció de les variables

- **dieFBX:** emmagatzema el *prefab* que s'instancia quan l'enemic perd la vida.
- **playerInRange:** indica quan el jugador es troba a rang de l'enemic.
- **dieEnemy:** indica quan l'enemic mort.
- **bulletPrefab:** emmagatzema el *prefab* que s'instancia quan l'enemic realitza un atac sobre el jugador.
- **canAttack:** booleà que indica quan es permet atacar al jugador.
- **timeBetweenAttacks:** variable que indica la freqüència d'atac del enemic.
- **attackDamage:** referencia la intensitat de l'atac especial del enemic.
- **visionRadius:** marca la distància mínima de detecció per a que l'enemic avanci cap al jugador.
- **damageSpecialAtt:** quantifica la intensitat de l'atac especial del jugador.
- **health:** indica la salut de l'enemic.

Descripció dels mètodes principals

- **OnTriggerEnter():** mètode s'encarrega de reduir la salut de l'enemic quan es realitza un atac sobre ell.
- **CanAttackPlayer():** aquest mètode indica quan la distància del jugador respecte l'enemic és més petita que la distància mínima de detecció.
- **AttackSpecial():** mètode que s'encarrega instanciar l'atac especial de l'enemic quan es donen les condicions.
- **SoundControl():** mètode que gestiona els sons de l'enemic.
- **AnimControl():** mètode que gestiona les animacions de l'enemic.
- **Die():** mètode que es crida quan la salut de l'enemic és igual o menor a zero.

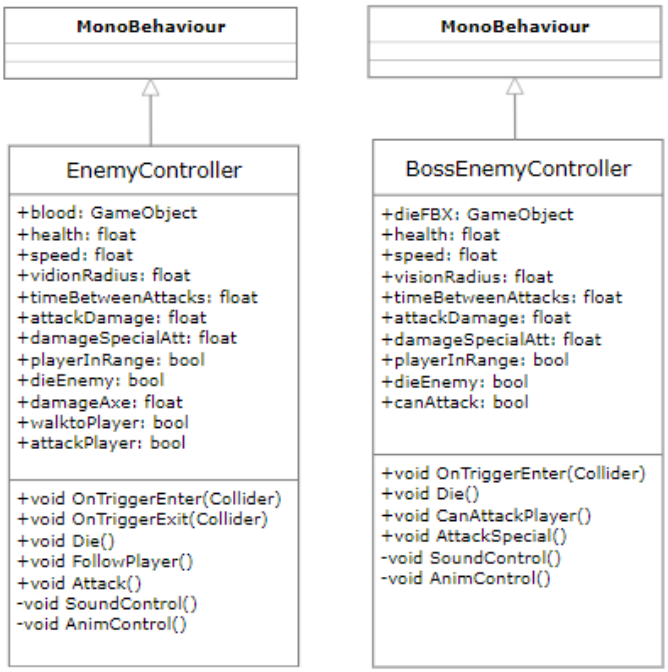


Figura 5.40 - UML dels *Scripts EnemyController i BossEnemyController.*

5.3. Objectes del mapa

Un dels principals objectius d'un joc estil RPG és la recerca d'objectes per l'escenari. Aquest objectes es troben repartits pel mapa i permeten interaccionar amb el jugador per tal de realitzar una acció en concret. Es distingiran aquests objectes segons la seva funció, és a dir, segons l'acció que tindrà sobre el jugador o bé com reaccionarà aquest amb els altres objectes.

L'objectiu d'afegir els objectes interactius és donar vida a l'escenari, donant lloc a una recerca constant d'objectes al voltant del mapa per permetre al jugador seguir avançant. S'han dissenyat els objectes de manera que tots siguin imprescindibles per progressar, ja sigui per desbloquejar algun camí o per derrotar enemics especials.

Aquests objectes són localitzats pel jugador mitjançant un detector anomenat *Interaction Zone,* que s'acciona quan l'usuari entra dins del radi d'acció del objecte. Consta d'un *Box Collider trigger* en forma de cub posicionat al davant del model simulant les mans del jugador. Quan el jugador interfereixi amb el *collider* de l'objecte, es permetrà realitzar algun tipus d'acció. Segons si l'objecte permet ser agafat pel jugador o no, es classificaran amb dos tipus.

Figura 5.41 *- Intersecció entre els colliders d'un objecte i el jugador*

Tots els objectes que permeten interacció amb el jugador, tenen implementat l'*script PickableObject* que habilita al objecte a modificar el seu comportament en funció de l'acció de l'usuari. S'encarrega també d'identificar l'objecte que es troba al davant de la zona d'interacció del jugador *Interaction Zone* i realitzar les accions oportunes segons la nominació de l'objecte.

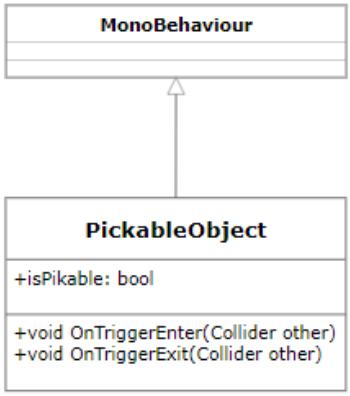


Figura 5.42 *- UML del Script PickableObject.*

5.3.1. Objectes per agafar

Són aquells elements que el jugador guardarà al seu inventari per realitzar una acció posterior. Tots els models utilitzats per conformar l'objecte són *assets* de lliure disposició i s'han descarregat de *l'Asset Store*.

5.3.1.1. Arma principal

Per implementar el sistema d'atac del jugador s'ha optat per utilitzar una atxa com a element per realitzar atacs als enemics.

Figura 5.43 *- Model 3D de l'arma principal.*

Aquest objecte es trobarà al principi del mapa i serà imprescindible obtenir-lo per seguir amb el correcte desenvolupament del joc.

Figura 5.44 *- Localització de l'arma principal a l'escenari.*

Per convertir el model de l'atxa en una arma funcional s'implementa el component *CapsuleCollider* i s'ajusta a la mida de la fulla per esdevenir-la un col·lisionador.

Aquest *collider* s'ha configurat mitjançant *l'script CharController* per a que només s'activi quan el jugador es troba en la posició d'atac (*AxeMotion*). Serà el *trigger* que detectarà l'enemic quan intercedeixi amb el propi *collider* de l'enemic.

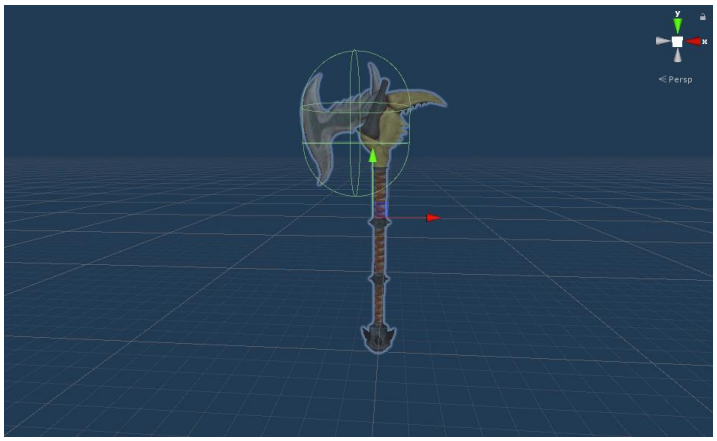


Figura 5.45 *- Collider de l'arma principal.*

Juntament amb el component *CapsuleCollider,* també conté els components *Transfrom* per ajustar la posició, rotació i escala del objecte, i l'Animator que permet incorporar animacions al model.

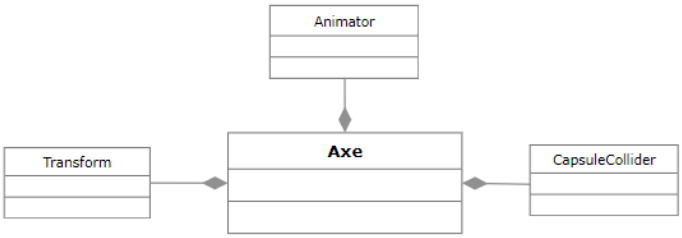


Figura 5.46 *- Components de l'arma principal.*

5.3.1.2. Claus

Per dificultar recorregut del jugador, s'han implementat unes claus que permetran desbloquejar el camí correcte. Aquest objectes es trobaran al terra de l'escenari, obligant al jugador a fixar-se amb la resta d'objectes repartits pel mapa. Les claus donaran la possibilitat de interactuar amb altres objectes i per tant, seran imprescindibles per avançar en la direcció correcta i superar el nivell.

Figura 5.47 *- Model 3D d'una clau*

Per facilitar la recerca d'aquest objecte s'ha implementat una llum focal intermitent que permet detectar-les a una distància major que la resta. També, s'adverteix al jugador de la necessitat de trobar un parell de claus per desbloquejar una porta mitjançant un quadre de diàleg, donant a entendre la importància d'aquests objectes.

Figura 5.48 *- Localització de les claus a l'escenari.*

5.3.1.3. Tresors

S'han incorporat unes caixes, que simulen ser tresors, per permetre al jugador progressar pel nivell. Aquests objectes tenen la peculiaritat de poder ser transportats d'un lloc a un altre de l'escenari, amb la finalitat de dipositar-los al lloc correcte per desbloquejar algun camí. No obstant, aquests objectes tenen efecte sempre i quan es col·loquen en les plataformes corresponents.

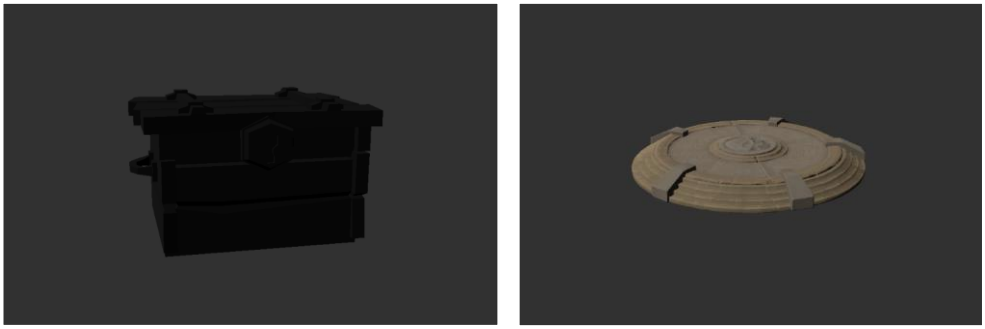


Figura 5.49 *- Models 3D dels tresors i plataformes.*

La detecció d'un tresor en una plataforma en concret es basa amb el sistema de intersecció entre dos *colliders*. En aquest cas, les plataformes contenen el disparador *trigger* que actua quan l'objecte amb el *tag* corresponent entra dins el seu radi d'acció.

Es distingiran tres tipus de tresors i tres tipus de plataformes segons el seu color: vermell, blau i verd.

Figura 5.50 *- Localització dels tresors a l'escenari.*

Figura 5.51 *- Localització de les plataformes a l'escenari.*

5.3.2. Objectes d'interacció

Aquests tipus d'objectes es caracteritzen per la funció que desenvolupen quan l'usuari interactua amb ells. Aquests modificaran les característiques del jugador i permetran travessar camins que prèviament estaven bloquejats.

El sistema de control es basa amb l'activació de variables booleanes a partir de *Scripts.* Amb el mateix sistema que els objectes anteriors, basat amb la intersecció de dos *colliders*, es dota al jugador la capacitat de variar la condició booleana d'aquesta variable i així, realitzar unes accions determinades.

5.3.2.1. Pous

Per variar les condicions del jugador durant la partida s'utilitzaran aquests tipus d'objectes, implementant el següent model a llocs estratègics del escenari.

Figura 5.52 *- Model 3D del pou.*

Per incorporar habilitats noves al jugador durant el transcurs del joc s'ha optat per utilitzar aquests tipus d'objectes. Com s'ha comentat en capítols anteriors, el jugador es capaç de llençar una bola d'energia en qualsevol direcció sempre hi quan tingui prou *stamina.* Per aconseguir aquesta habilitat es necessari interactuar amb aquest objecte simulant que el jugador beu d'aquest pou i adquireix aquest poder.

Figura 5.53 *- Localització del pou d'energia a l'escenari.*

Per altra banda, també s'ha utilitzat aquest model per variar la característica de salut del jugador regenerant la seva vida al interactuar amb els pous amb el foc de color verd. Quan s'interacciona amb l'objecte, es crida a la funció "*HealPlayer()*" que conté el *Script* "*PlayerHealth*" que incrementar la vida del jugador un 100%.

Figura 5.54 *- Localització del pou de salut a l'escenari.*

5.3.2.2. Portals

Els portals són objectes d'interacció que realitzaran la seva funció si el jugador conté en el seu inventari uns objectes que s'aconsegueixen arreu del mapa, o bé aconsegueix superar un puzle.

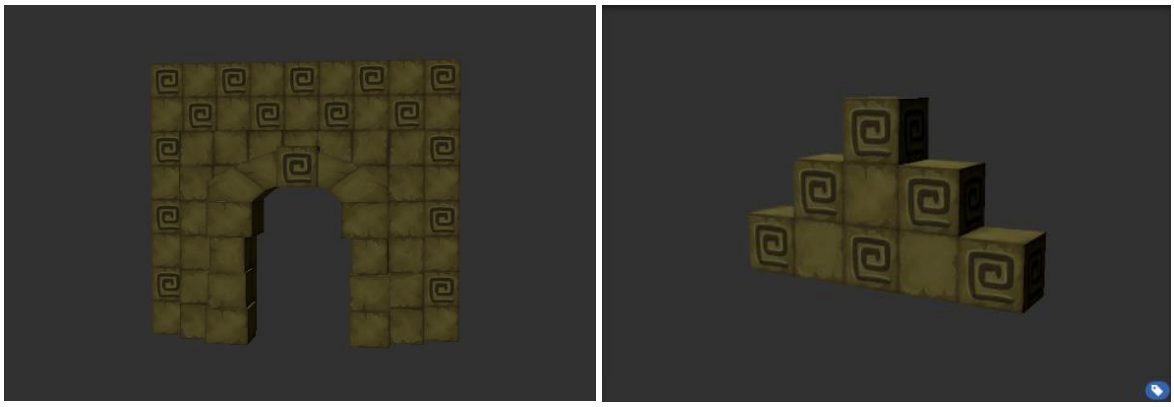


Figura 5.55 *- Models 3D del portal principal i secundari.*

Els objectes necessaris són les claus, i realitzaran la funció de desbloquejar el portal principal. Quan el jugador obtingui les dues claus, canviarà el valor lògic d'una variable booleana que permetrà obrir el portal quan l'usuari interaccioni amb ell.

Figura 5.56 *- Localització del portal principal a l'escenari.*

El portal està format per dos elements: l'estructura i la porta d'entrada. Cada element exerceix una funció concreta per que el sistema funcioni correctament. L'estructura conté el component *BoxCollider* que l'identifica amb el *tag* "Portal" distingint-lo de la resta d'objectes. La porta d'entrada conté el component *Animator* i té associada una animació que s'executa quan el jugador interacciona amb l'objecte. El component BoxCollider que té l'objecte que engloba els dos elements, és l'encarregat de permetre al jugador interaccionar amb l'objecte.

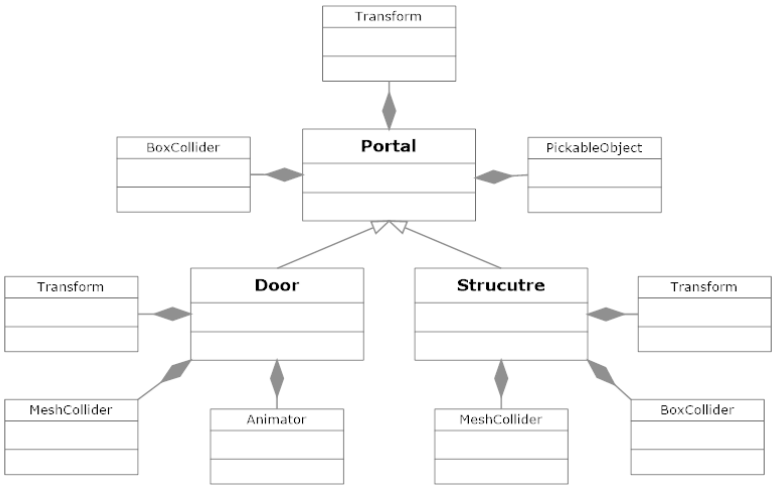


Figura 5.57 *- Components del portal principal.*

Per accedir a la zona següent s'ha de superar un puzle que desbloqueja portal seccundari. S'ha utilitzat una piràmide per evitar el progress del jugador.

Figura 5.58 *- Localització del portal secundari a l'escenari.*

Aquest portal es controlat per l'*script GateController* que s'enllaça amb el controlador de l'escena *SceneController*, encarregat de gestionar la solució de l'enigma. Quan l'*script* del portal secundari rep el valor lògic d'un booleà que confirma que s'ha superat el puzle, s'executa una animació amb el component *Animator* que desplaça la piràmide i desbloqueja el camí perquè el jugador pugui progressar en la història.

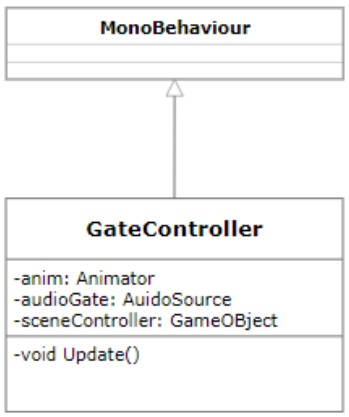


Figura 5.59 *- UML del Script GateController.*

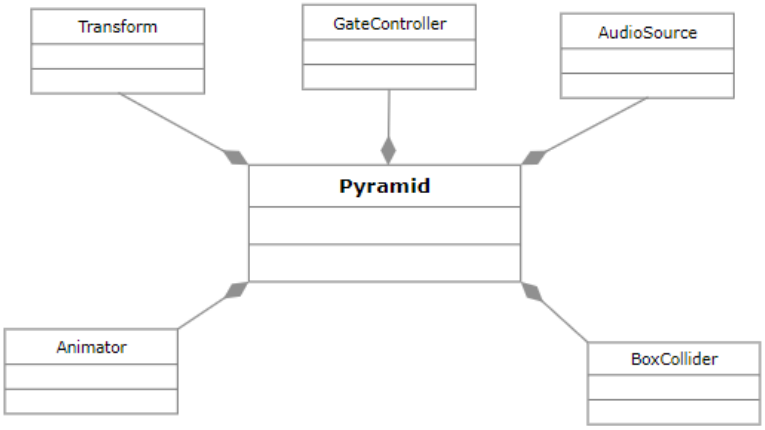


Figura 5.60 *- Components de la pirámide.*

5.3.2.3. Cubs del enigma

Una de les principals característiques d'un videojoc de rol és la presa de dedicions del personatge envers a una dificultat. Per intercedir en el trajecte del jugador, s'han incorporat tres cubs per formar un enigma.

Figura 5.61 *- Localització del tresor verd a l'escenari.*

L'enigma consistirà en interaccionar amb els objectes en l'orde correcte per desbloquejar un camí. La solució de l'enigma es basa amb l'ordre dels colors dels tresors prèviament localitzats al voltant del mapa, i es gestionada per l'script *SceneController.*

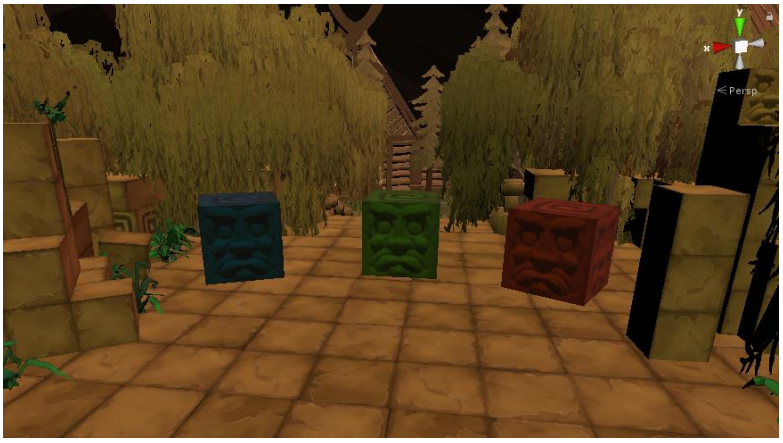


Figura 5.62 *- Localització del tresor verd a l'escenari.*

6. Escenaris

L'editor d'Unity 3D inclou una eina que permet crear un terreny amb un conjunt integrat de característiques personalitzades anomenada *Terrain*. L'editor es capaç d'ajustar l'altura o la aparença del seu paisatge, agregant arbres o diferents tipus de vegetació, així com modificar la forma del terreny. En aquest s'han afegit dos extensions addicionals amb les mateixes característiques que el principal per augmentar la capacitat de l'escenari.

Les principals característiques que s'han modificat per elaborar el disseny del terreny són:

- **Llargada i amplada:** s'ha ajustat el mida del terreny a 100 x 100 *(world units)*
- **Altura:** s'ha modificat l'altura del terreny per construir muntanyes i altres obstacles de l'escenari. També s'ha augmentat l'altura de tota la superfície una unitat per poder crear un riu que travessa verticalment el mapa.
- **Textura:** s'han afegit diferents textures per donar relleu al escenari i modificar la seva aparença.
- **Arbres i vegetació:** s'han importat *prefabs* que contenen models 3D d'arbres, pedres, plantes i altres tipus de vegetació per millorar l'aspecte de l'escenari.

6.1. Tipus d'escenari

S'han dissenyat dos escenaris per cadascuna de les modalitats possibles del videojoc. En primer lloc, s'ha creat un mapa per al mode un jugador de grans dimensions i un altre per al mode multijugador, de dimensions més reduïdes. Els escenaris s'adapten a les condicions del mode de joc, tant en la mida del mapa com en la distribució de zones i objectes.

6.1.1. Un jugador

S'ha dissenyat un escenari ampli i simètric amb les característiques que disposa l'eina descrita anteriorment. Consta de tres unitats de *Terrain* amb les mateixes dimensions distribuïts de manera horitzontal, és a dir, la unitat principal es troba el la posició (0,0) i les altres dues unitats es situen a les posicions (1,0) i (-1,0).

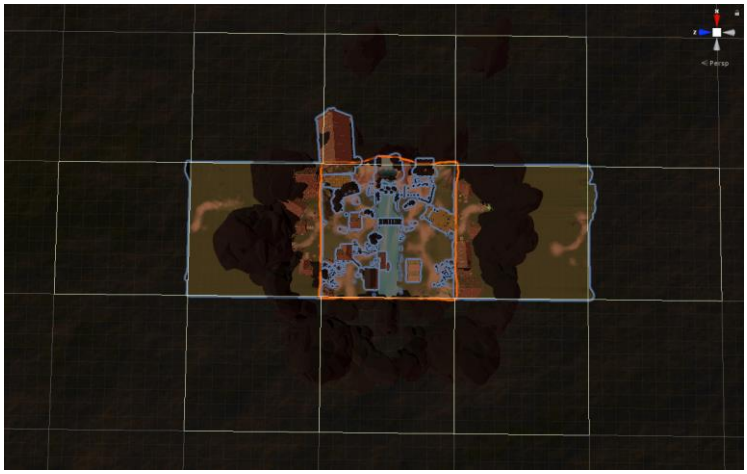


Figura 6.1 *- Mòduls del terreny de l'escenari principal.*

S'ha modificat l'altura del terreny dels voltants de l'escenari per crear muntanyes naturals que estableixen els límits del mapa. S'ha creat una cascada amb la mateixa eina i s'ha dissenyat un riu que separa l'escenari en dos parts per facilitar-ne la distribució de zones i el recorregut del jugador.

S'han utilitzat cinc tipus de textures per millorar l'aspecte de l'escenari i donar-li relleu. Les textures verdes simulen zones riques en vegetació i les verdes amb fulles s'utilitzen per a les zones on s'han implementat boscos o espais frondosos. Per marcar el recorregut del jugador facilitant així el seu progrés, es pinta el terra amb textures de color marró. També s'utilitzat la textura marró amb pedres per a les zones que simulen ser interiors de coves o espais rocosos. Per últim, s'ha pintat el fons del riu amb la textura blava que simula el color de l'aigua.

Figura 6.2 *- Textures del terreny.*

A més a més de la distinció de zones per textures, també s'han implementat *prefabs* de molts tipus de vegetació que milloren l'aspecte del escenari i ambienten al jugador segons la zona on es trobi. S'ha utilitzat aquesta eina per crear espais més frondosos o més àrids en funció del que es vulgui transmetre a l'usuari i permetent dividir l'escenari per zones.

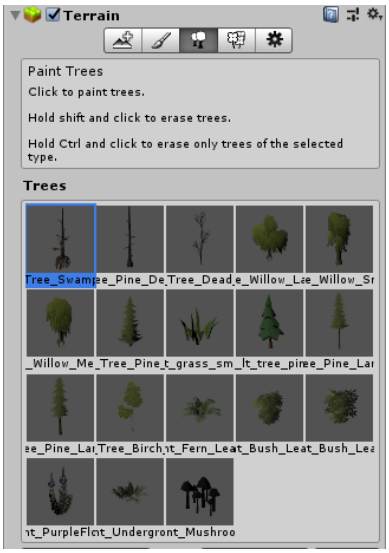


Figura 6.3 *- Models 3D de la vegetació del terreny.*

Amb les modificacions del terreny explicades s'ha dissenyat un escenari complet, ampli i amb un aspecte visual força atractiu.

Figura 6.4 *- Disseny del escenari principal del mode un jugador.*

6.1.2. Multijugador

L'escenari dissenyat per al mode multijugador és notablement més petit que el del mode d'un jugador. S'utilitzen els mateixos recursos per dissenyar-lo per tal de fer notar a l'usuari el lligam que manté amb la història principal. S'ha fet ús de les mateixes eines de disseny del escenari per incorporar vegetació i textures per distingir el mapa en zones d'interès.

Consta d'un escenari quadrat i simètric dividit per un riu de lava que separa les dues zones rellevants del mapa.

Figura 6.5 *- Disseny del escenari principal del mode multijugador.*

6.2. Disseny del nivell

En aquest apartat s'explicarà com s'ha dissenyat els nivells fent distincions per zones en funció de l'acció a desenvolupar dels personatges. Es farà un recorregut seguint l'orde de la història del mode d'un jugador explicant quines accions s'esperen del jugador en aquella zona. També es parlarà de les zones més rellevants de l'escenari creat per al mode multijugador.

6.2.1. Un jugador

Seguint l'ordre del transcurs de la història, ens trobem en primer lloc una zona àrida que destaca sobre la resta de l'escenari per ajudar al jugador a fixar-se amb aquell espai ja que és allí on s'aconsegueix la arma principal del personatge.

Figura 6.6 *- Zona 1 del mode un jugador.*

Seguidament ens trobem amb la zona d'aprenentatge, on el jugador es troba amb una sèrie d'obstacles que ha de superar amb l'ajuda d'un quadre de diàleg que l'indica com fer-ho. En aquest espai també s'ensenya al usuari ha realitzar maniobres d'atac i defensa.

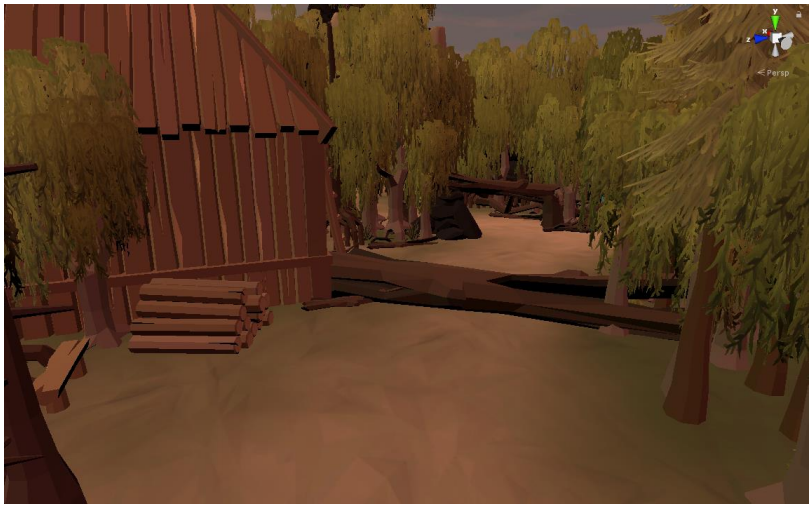


Figura 6.7 *- Zona 2 del mode un jugador.*

A continuació es troba una disjuntiva entre dos camins. Es imprescindible que el jugador s'adoni quin camí ha de seguir de primeres. Per facilitar-ho, un quadre de diàleg s'encarrega de advertir al jugador del camí que ha d'agafar. En aquest punt, s'aconsegueix l'atac especial interaccionant amb un pou (objecte d'interacció*)* i es mostra a l'usuari com fer-ho.

Figura 6.8 *- Zona 3 del mode un jugador.*

Després d'eliminar el primer enemic, el jugador creuarà un pont on es trobarà amb el següent objecte d'interacció que li permetrà recuperar salut. Tot seguit s'enfrontarà a més enemics que haurà de derrotar per progressar amb la història.

Figura 6.9 *- Zona 4 del mode un jugador.*

Seguidament el jugador arribarà a la zona on hi ha un dels portals (objecte d'interacció) on se li requerirà un parell de claus per poder accedir-hi. Aquestes es trobaran en llocs estratègics de l'escena i quan l'usuari les tingui al seu inventari, haurà de tornar a la zona per desbloquejar el portal i avançar al següent desafiament.

Figura 6.10 *- Zona 5 del mode un jugador.*

En aquesta zona es trobarà el primer tresor (objecte per agafar) que el jugador haurà d'agafar i endur-se'l camí endavant. Primer haurà de superar els enemics que s'oposen al recorregut.

Figura 6.11 *- Zona 6 del mode un jugador.*

En aquest punt, el jugador haurà de superar els obstacles per arribar fins al riu i travessar-lo per les pedres. Prèviament s'adverteix de la perillositat de caure a l'aigua, ja que si ho fas perds tota la salut.

Figura 6.12 *- Zona 7 del mode un jugador.*

Quan es travessa el riu, s'arriba a la zona més rellevant de tot l'escenari. Es tracta de l'espai on es troben les plataformes de colors que es relacionen amb els tresors. En aquest punt, el jugador trobarà en disposició de col·locar el primer tresor de color vermell al lloc que li correspon. També es la zona on es trobarà una de les dues claus que es necessita per desbloquejar el portal descrit a la zona 5.

Figura 6.13 *- Zona 8 del mode un jugador.*

A la següent zona es troba el segon tresor. El jugador haurà de derrotar els enemics que protegeixen aquest tresor i portar-lo a la zona 8 de nou per ficar-lo a la plataforma corresponent. En aquesta zona es troba també la segona clau per desbloquejar el portal

Figura 6.14 *- Zona 9 del mode un jugador.*

En aquesta zona s'accedeix després de desbloquejar el portal de la zona 5. Es descobreix un enigma basat amb tres cubs de colors que desbloquejaran un camí quan s'interaccioni amb ells amb l'orde correcte. La solució de l'enigma donarà pas a la zona 11.

Figura 6.15 *- Zona 10 del mode un jugador.*

Aquesta zona és considera d'importància ja que conté el tercer i últim tresor necessari per desbloquejar el final de la història. Es situa en un petit poblat adjunt a l'escenari principal on hi ha una sèrie d'enemics protegint el tresor. Es pot considerar com una zona simètrica a la zona 9.

Figura 6.16 *- Zona 11 del mode un jugador.*

Finalment, amb la obtenció de l'ultim tresor, el jugador haurà de passar per les zones 7, 8 i 10 fins arribar a la zona final. Quan el jugador identifiqui les tres plataformes de la zona 8 amb els tresors obtinguts durant el camí i les coloqui al lloc corresponent, es desbloquejarà una porta per accedira la zona 12. En aquest tram es trobarà l'enemic final i es produirà el desenllaç de la història.

Figura 6.17 *- Zona 12 del mode un jugador.*

6.2.2. Multijugador

L'escenari del mode multijugador s'ha dissenyat de forma simètrica de manera que es podran distingir tres zones d'interès. La primera zona mostra el lloc d'aparició del personatge masculí Harald i serà en aquest punt on iniciarà la partida.

Figura 6.18 *- Zona 1 del mode multijugador.*

La següent zona és la complementaria a l'anterior ja que és el punt d'aparició del personatge femení Thyra. Les dues zones són inversament simètriques i és distingiran pels portals de grans dimensions com a punt de partida.

Figura 6.19 *- Zona 2 del mode multijugador.*

La darrera zona es pot considerar com el punt de conflicte entre els dos jugadors, ja que uneix les dues zones i permetrà als dos jugadors interaccionar entre ells.

Figura 6.20 *- Zona 3 del mode multijugador.*

6.3. Control de l'escena

Per governar sobre les accions i esdeveniments que succeeixen durant el transcurs de la història, s'ha creat un *GameObject* buit anomenat Scene Controller. Aquest té implementat l'*script SceneController* que s'encarrega de controlar aquestes accions.

SceneController accedeix a les variables de decisió d'altres scripts per controlar els esdeveniments de l'escena. Les funcions que desenvolupa són:

- Mostra el text d'interacció del jugador quan aquest li permet.
- Recull informació sobre la interacció del jugador amb els objectes de l'escenari.
- Té el control de l'obertura de portes segons els esdeveniments succeïts.
- Gestiona la gran majoria d'efectes especials de l'escena.
- S'encarrega de gestionar la solució de l'enigma.
- Manté el control de l'aparició dels enemics a l'escena.
- Es gestionen els sons que es reprodueixen a l'escena quan hi ha algun succés.

L'objecte Scene Controller també conté l'*script DialogueController* encarregat de gestionar el sistema de diàlegs entre el jugador i la màquina.

A més, s'ha afegit el component *AudioSource* ja que és imprescindible per manipular els clips de so i reproduir-los .

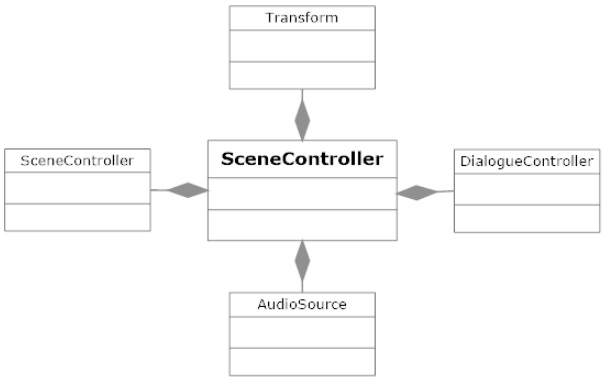


Figura 6.21 - Components del *Script SceneController.*

Descripció de les variables

- **youWin:** emmagatzema l'objecte que apareix quan el jugador guanya la partida.
- **gameOver:** emmagatzema l'objecte que apareix quan el jugador perd la partida.
- **explosionVFX:** emmagatzema el *prefab* que s'instancia quan s'assoleix un objectiu.
- **explosionPosition:** referència la posició de l'escenari on es té que instanciar l'explosió.
- **enemyBoss1, enemyBoss2:** emmagatzemen els *prefabs* dels enemics que apareiexen quan s'assoleix un objectiu.
- **boss1Respawn, boss2Respawn:** referència la posició de l'escenari on es tenen que instanciar els enemics.
- **redPlatform:** accedeix a la plataforma vermella del escenari.
- **bluePlatform:** accedeix a la plataforma blava del escenari.
- **greenPlatform:** accedeix a la plataforma verda del escenari.
- **allBoxes:** indica si els tresors s'han col·locat en el lloc corresponent.
- **keyInInventary:** indica si el jugador conté en el seu inventari les dues claus.
- **boxInRed, boxInBlue, boxInGreen:** indica si el tresor es troba a la plataforma corresponent.
- **boxBlueRed:** indica si els tresors blau i vermell es troben a les seves respectives plataformes.
- **correctWay1, correctWay2:** indica si el procés de l'enigma és correcte.
- **activeAllBoxes:** indica quan tots els tresors es troben al lloc corresponent.
- **openGate:** indica quan es permet desbloquejar el portal de l'enigma.
- **oneKeyInInventary:** indica si el jugador conté almenys una clau al seu inventari.

Descripció dels mètodes principals

- **Controller():** mètode que realitza les accions de l'escena en funció de les variables de decisió.
- **EnemysRespaw():** mètode que gestiona l'aparició d'enemics per l'escena.
- **SceneAudio():** mètode que governa els sons de l'escena.

A l'annex A3 (Codis font) es mostra el contingut de *l'script SceneController*.

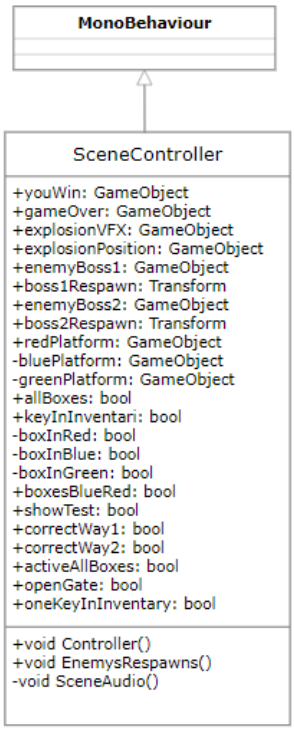


Figura 6.22 - UML del *Script SceneController.*

7. Interfície gràfica

En un videojoc d'estil RPG la interfície gràfica és un dels elements imprescindibles que milloren l'aspecte visual i contribueixen en la immersió de la història. La formen tots els tipus de dispositius o informacions que apareixen en la pantalla que permeten al jugador tenir un control i seguiment dels elements del videojoc. Actua com a punt d'interacció entre l'espai físic amb el digital, és a dir, entre l'usuari i el videojoc.

És important que la interfície gràfica estigui a l'altura del videojoc, és a dir, ha de transmetre una certa continuïtat entre l'estètica i la història que s'està explicant, així com l'ambientació de l'escena. En la majoria de videojocs el disseny de la interfície gràfica es basa amb la creació de personatges, configuració d'habilitats o gestió d'inventari. El disseny de la interfície ha de ser simple i intuïtiva ja que l'usuari invertirà temps de joc interactuant amb aquests elements i per tant, la consistència d'aquests milloraran l'experiència del jugador.

En aquest projecte, s'ha implementat un conjunt de elements que formen la interfície gràfica digital del videojoc. S'han dissenyat els elements que mostren informació del sistema i del joc relacionada amb característiques o habilitats del personatge anomenada HUD *(heads-up display).* També s'han dissenyat els elements que mostren i permeten al jugador manipular informació.

7.1. HUD

Per mantenir un seguiment de la informació del jugador s'han dissenyat un conjunt d'elements que formen part del HUD situat dalt a l'esquerra de la pantalla. Consta d'una barra de color verd que indica la salut actual del jugador, una barra de color blau que representa la *stamina* del atac especial i un conjunt d'icones que donen informació al jugador d'habilitats i objectes que ha aconseguit fins al moment. També es mostra una imatge del jugador per relacionar aquestes habilitats amb el personatge que s'està fent ús. Aquest element s'ha dissenyat amb la visió de poder incorporar una segon personatge i permetre al jugador seleccionar-los abans d'iniciar la història.

Per millorar l'aspecte del HUD s'han dissenyat uns efectes visuals que s'activen quan el jugador recupera salut quan interacciona amb l'objecte corresponen o li resta salut degut a l'atac d'un enemic. També s'ha fet ús d'un *asset* de l'Asset Store que conté multitud d'elements per crear una interfície gràfica. S'han implementat molts dissenys d'aquest *asset* per confeccionar altres components del GUI *(graphical user intereface),* en aquest cas, s'ha optat per un disseny simple que dona relleu i contorn als elements descrits anteriorment i també contribueix en millorar l'estètica del HUD.

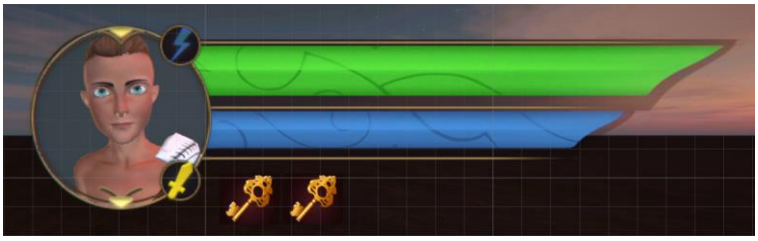


Figura 7.1 *- Disseny del HUD del jugador.*

A continuació s'explica com s'han dissenyat els components del HUD i quins són els controladors que gestionen les funcions d'aquests elements.

7.1.1. Barra de salut (health bar)

Aquest element indica la salut del jugador a temps real. Es tracta d'un *slider* horitzontal de color verd disposada sobre un marc fosc que dóna contorn al disseny. El control d'aquest element es gestionat per l'*script PlayerHealth* a partir de les variables enteres (int) *startingHealth* i *currentHealth.* La primera indica el valor inicial de la salut del jugador, inicialitzada en 100 unitats; i la segona emmagatzema el valor actual. *CurrentHealth* és la variable que ajusta el valor del *slider* anomenat *healthSlider* i s'actualitza quan el controlador de l'enemic crida a la funció *TakeDamage()* indicant la quantitat de salut a restar.

```
public void TakeDamage(int amount)
{
    if (!player.Block)
    {
        damaged = true;
        currentHealth -= amount;
        healthSlider.value = currentHealth;
```

Figura 7.2 *- Funció TakeDamage del script PlayerHealth.*

7.1.2. Barra d'energia (mana bar)

La barra d'energia o més coneguda com barra de manà s'encarrega de gestionar la *stamina* de l'atac especial del jugador. Consta d'una imatge horitzontal de tipus *Filled* que s'escala entre 0 i 1 en funció del seu valor. Aquest element s'ha incorporat per evitar que l'usuari abusi d'aquesta habilitat i que, per tant, utilitzi totes les funcions que té el personatge. Per gestionar aquest element es fa ús de les variables flotants *myMana* i *currentMana* que emmagatzemen el valor màxim i actual de l'energia disponible. També s'utilitza la variable booleana *canMana* per decidir si el jugador té prou energia per realitzar un atac especial. Cada cop que es llança una bola d'energia es crida a la funció

ReduceMana() que ajusta la imatge a 0 i incrementa progressivament el seu valor fins al màxim amb la funció *Update().*

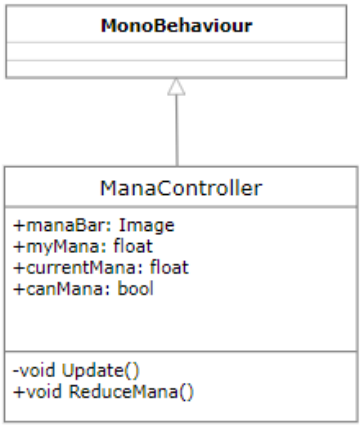


Figura 7.3 *- UML del script ManaController.*

7.1.3. Icones del HUD

Es distingeixen dos tipus d'icones, els que fan la funció d'inventari mostrant els objectes recollits fins al moment, i els que il·lustren les habilitats del jugador. Del primer tipus, s'han dissenyat dues icones simbolitzant unes claus que donen constància dels objectes que s'han trobat arreu de l'escenari. Del segon tipus, s'han implementat dues imatges al voltant de la imatge de l'avatar del jugador que indiquen les habilitats obtingudes fins al moment. S'ha dissenyat una icona amb una espasa de color groc que apareix quan el jugador obté l'arma principal. També s'ha dissenyat una icona amb un raig de color blau que es manifesta quan el jugador obté habilitat de llançar un atac especial.

Tots dos tipus d'icones són gestionats de la mateixa manera. A partir de l'*script PickUpObjects* s'enllaça les imatges de les icones emmagatzemades en *GameObjects* que s'inicien desactivades amb la funció *gameObject.SetActive(false).* Quan el jugador interactua amb els objectes descrits en l'anterior paràgraf, s'activen els GameObjects que emmagatzemen els icones amb la mateixa funció *gameObject.SetActive(true)* segons amb quin objecte s'hagi interaccionat.

7.1.4. Efectes visuals del HUD

Consta de dues imatges de color verd i vermell que es situen a la mateix posició i amb la mateixa mida que la barra de salut *(healthSlider)* per indicar a l'usuari quan l'avatar perd o recupera salut provocat per factors externs, com l'atac d'un enemic o interaccionar amb un pou de salut. Aquest conjunt d'imatges contribueixen a produir un efecte visual cada cop que la barra de vida canvia de valor, però també formen part del HUD ja que es mostra a la part superior esquerra juntament amb la resta d'elements.

Es gestiona a partir de l'*script PlayerHealth* que conté les variables *damageImage* i *healingImage.* Aquestes variables emmagatzemen imatges que s'activen quan les variables booleanes *damaged* o *healing* canvien el seu valor lògic respectivament. Les imatges es configuren amb el mateix *script,* s'ajusta el color a partir del codi de colors d'Unity3D i la velocitat d'aparició de la imatge. El color vermell correspon al codi (1,0,0,1) i el color verd al codi (0,1,0,1) i s'emmagatzemen a les variables *flashColourDamage* i *flashColourHeal*. La velocitat en que esdevé de la transparència al color configurat es gestiona amb la variable flotant *flashSpeed.* La funció Update() es l'encarregada de manipular aquestes variables i executa les funcions segons el valor de les variables booleanes.

7.2. Components UI

Hi ha altres elements que formen part de la interfície gràfica del jugador (GUI) però que són independents als components del HUD. Aquests elements s'han dissenyat per a donar la informació necessària per a que l'usuari interactuï amb fluïdesa durant el transcurs del videojoc. Guiaran al jugador per a que pugui accionar i recórrer la narrativa correctament de manera directa i intuïtiva.

S'ha implementat un text que es mostra quan l'usuari es situa al davant d'un objecte amb capacitat d'interacció i un altre conjunt de textos que indiquen quan el jugador aconsegueix la victòria o derrota en qualsevol instant del videojoc. Per altra banda, s'ha dissenyat un sistema de diàlegs que permet a l'usuari mantenir contacte amb la màquina o equip, és a dir, s'utilitza com a punt d'interacció entre els dos subjectes per millorar el funcionament i per fer més efectiu el control de la màquina des de la interacció amb l'humà. Per últim, s'ha configurat una imatge que actua com a punt de referència per realitzar un atac principal o especial.

7.2.1. Text interacció

S'ha creat un *GameObject* anomenat InfoPlayer que en conté el component *Text* que permet introduir un text i configurar paràmetres com la font, mida o estil. Aquest element s'activa quan el jugador s'apropa a un objecte que conté l'*script PickableObject* amb la variable booleana *isPickable* activa. Llavors es mostra el text "Prem 'F' per interactuar" per indicar a l'usuari que aquell element permet interacció amb el jugador.

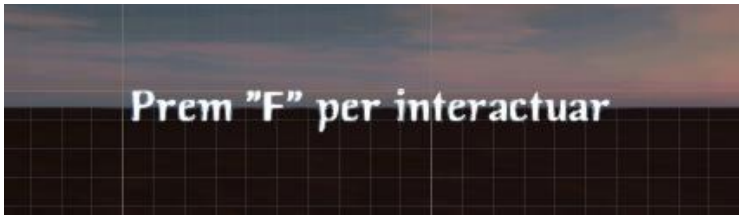


Figura 7.4 *- InfoPlayer amb el text d'interacció.*

Per controlar aquest objecte s'utilitzen els *scripts PickUpObjects* i *SceneController.* El primer conté la variable booleana *showPickUpText* que s'activa quan la zona d'interacció del jugador (Interaction Zone) intersedeix amb el *collider* d'un objecte amb la condició *isPickable* amb valor *true.* Aquesta variable es recupera a l'*script SceneController* que és qui gestiona el *GameObject* InfoPlayer. La manera d'activar i desactivar l'objecte que conté el text és amb la funció *gameObject.SetActive().*

7.2.2. Text victòria/derrota

Per emfatitzar quan el jugador canvia als estats de victòria i mort s'ha optat per dissenyar un text animat que s'invoca quan es donen aquestes condicions. S'han creat dos *GameObjects* anomenats YouWin i GameOver amb els components *Text,* per informar de l'estat al jugador, i *Animator* per gestionar les animacions a partir d'un controlador creat prèviament.

Figura 7.5 *- Objecte YouWin actiu (in Game).*

L'*script PickUpObjects* conté la variable booleana *gameFinish* que retorna verdader quan s'interacciona amb l'objecte final del mapa ("BoxAztec") i que permet donar les condicions per a que es mostri l'objecte YouWin. Per a que es mostri GameOver s'ha d'actuar sobre la variable de tipus booleà *youWin* emmagatzemada a l'*script PlayerHealth*, que canvia de valor quan es crida a la funció *Dead(). SceneController* és *l'script* que accedeix a les variables booleanes i activa els *GameObject*s amb la funció *SetActive()* quan aquestes són verdaderes.

7.2.3. Diàlegs

Per aconseguir establir una conversació entre l'usuari i la màquina s'ha dissenyat un sistema de diàlegs que permet al programador donar informació al jugador. Costa de dos *GameObjects* anomenats NameText i Dialogue amb un component *Text* afegit que s'encarreguen de mostra en pantalla el text que es configuri prèviament.

Figura 7.6 *- Sistema de diàlegs (in Game).*

El funcionament del sistema de diàlegs es basa amb la col·lisió entre dos *colliders* essent un d'ells de condició *trigger*. En aquest cas es crearà un objecte amb el component *BoxCollider* i el paràmetre *trigger* activat per detectar quan el *collider* del jugador entra dins el seu espai. No obstant, serà l'objecte SceneController qui contingui els *scripts DialogueController* i *Dialogue* que gestionaran el funcionament del sistema.

DialogueController recull els textos dels *GameObjects* NameText i Dialogue, el component *Animator* per incorporar animacions per a l'entrada i sortida del text i així millorar l'aspecte visual, i una variable privada formada per un conjunt de cadenes de caràcters (*strings*) anomenat *sentences. Dialogue* és la classe que emmagatzemarà el conjunt de caràcters que es voldrà mostra en pantalla i actua quan *DialogueTrigger* crida a la funció StartDialogue(Dialogue).

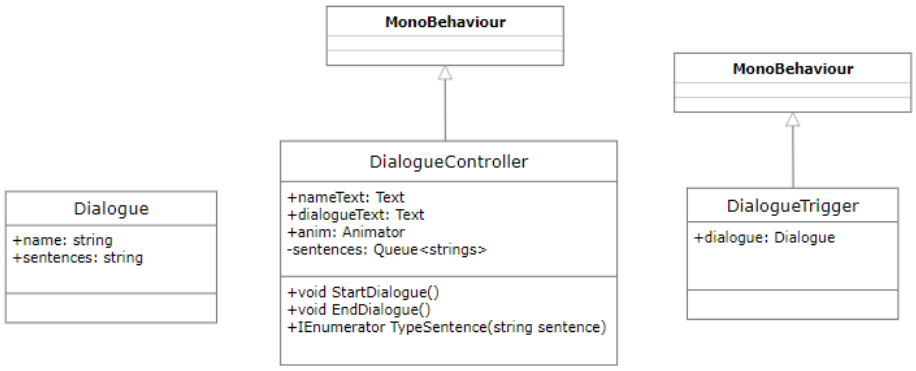


Figura 7.7 *- UML dels Scripts Dialgoue, DialogueController i DialogueTrigger.*

Aquesta funció es cridada quan s'executa *OnTriggerEnter()* i el *collider* que entra en contacte amb l'objecte que conté *DialogueTrigger* és el jugador. Quan aquest surt de la zona de contacte s'executa la funció *OnTriggerExit()* que crida a la funció *EndDialogue()* per finalitzar la conversa. Aquest objecte s'ha creat per a tots i cadascun dels diàlegs que es vol mostrar en cada zona, és a dir, es repartiran arreu de l'escena per a que el jugador entre en contacte amb ells a mesura que progressa la història.

S'han dissenyat 35 *GameObjects* que contenen *DialogueTrigger* amb els components *BoxCollider* ajustat en funció de la geometria de la zona, i 4 més que criden a la funció *StartDialogue()* en funció del valor d'un booleà, i finalitzen la conversa cridant a la funció EndDialogue() mitjançant una variable que emmagatzema el temps transcorregut des de l'inici de la conversa fins el límit estipulat pel programador. Entre tots aquests elements s'ha desenvolupat un sistema de diàleg robust i efectiu.

Figura 7.8 *- Triggers del sistema de diàlegs.*

7.2.4. Punt de mira (SMC)

Aquesta imatge actua com a punt de referencia per dirigir els atacs cap a una direcció concreta. S'ha afegit un *GameObject* amb el component *Canvas* al objecte que conté el personatge principal per facilitar-ne el seu control. Al *Canvas* s'ha afegit una imatge d'un model semblant a una creu per simular un punt de mira per ajudar al jugador a situar el centre de la pantalla i així utilitzar-lo com a referència per dirigir els atacs.

El control d'aquest element és molt simple. S'encarrega *CharController* de activa i desactiva aquest *GameObject* quan es donin les condicions. S'ha configurat de manera que quan la posició d'atac del jugador estigui activa (AxeMotion) s'activi el *Canvas* que permet visualitzar el punt de mira. Aquesta acció es du a terme amb la funció *gameObject.SetActive()* de la mateixa manera que es desactiva aquest *GameObject* quan el jugador retorna a qualsevol de la resta d'estats.

7.3. Menus i navegació

Els menús són un component imprescindible en gairebé tots els estils de videojoc. Tenen la funcionalitat de representar opcions que permeten al jugador manipular informació. Les opcions que es mostren s'utilitzen com a *inputs* per a que el jugador les manipuli i sigui capaç de navegar per les escenes del joc.

S'ha de tenir en compte que per a que l'experiència del jugador sigui completa, el videojoc ha de mantenir un contacte constant amb l'usuari per informar-li de totes les situacions que s'estan donant. La incertesa empitjora greument la experiència i mostra les carències i la falta de robustesa del projecte. Ressaltar un botó al passar pel damunt (*hover*), mostra un progrés mentre s'està carregant la següent escena o incorporar sons quan es prem alguna opció són alguns elements que contribueixen en la consistència del videojoc donant un *feedback* constant de les accions que s'estan realitzant.

Els menús i les escenes de navegació d'aquest videojoc s'han dissenyat a partir de *GameObjects* amb el component *Button* per a que aquest objecte tingui la mateixa funcionalitat que un botó. Els botons són entrades (*inputs*) dissenyades per a que l'usuari interaccioni amb ells i així permetre manipular informació i navegar per les diferents opcions que ofereix el videojoc. Els paràmetres que s'han modificat per dissenyar els botons que es mostren al menú són:

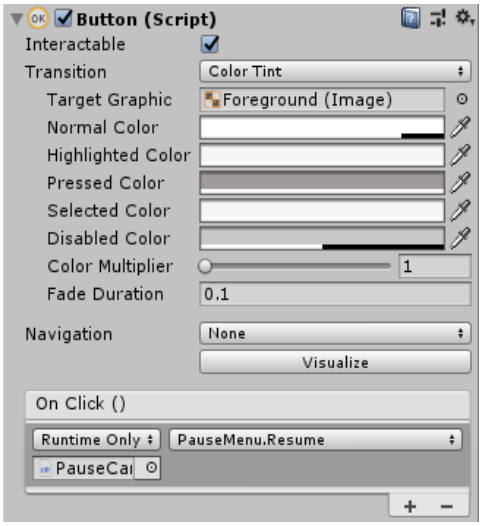


Figura 7.9 *- UML del Script PauseMenu*

- **Interactable**: aquesta opció decideix si el botó permet interacció amb l'usuari o no.
- **Normal Color**: és el color per defecte que mostra el botó. En aquest cas, s'ha optat per incorporar una imatge prefabricada provinent d'un *asset* amb color i relleu. Per tant, no s'ha modificat el color del botó, però s'ha reduït la transparència d'aquest per crear l'efecte *hover*.
- **Highlighted Color**: és el color que es mostra quan el punter es situa damunt del botó. S'ha configurat totalment opac per crear la transició amb *Normal Color.*
- **Pressed Color**: és el color que apareix quan es prem el botó en qüestió. No s'ha modificat l'opacitat del botó però s'ha variat el color de la imatge.
- **OnClick():** és la funció que executa les accions que s'enllacen al interaccionar amb el botó.

L'*script* que s'enllaça amb el botons per canviar d'escena segons el botó que s'hagi seleccionat és *LevelLoader.* Amb la llibreria importada *SceneManagment* es permet governar sobre els canvis d'estat del videojoc, ja sigui per canviar d'escena o bé per sortir de l'aplicació. La funció *LoadLevel()* permet carregar una escena introduint l'índex d'aquesta en funció de la seva posició respecte la resta d'escenes, per exemple, el menú principal correspon a l'escena amb índex 0 i el mode d'un jugador correspon a l'índex 1. Per altra banda, també es gestiona l'aparició de la pantalla de càrrega on es calcula el progrés de la càrrega amb la funció *LoadAsynchronously()*.

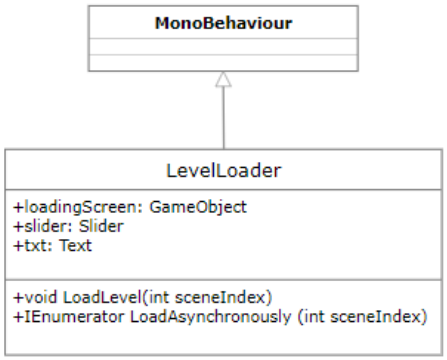


Figura 7.10 *- UML del Script LevelLoader*

En aquest videojoc, s'ha dissenyat un menú principal dinàmic des d'on s'accedeix als dos modes de joc disponibles. També s'ha afegit un botó per desplegar el menú d'opcions que permet regular el volum de la musica del joc. Per últim, un botó lleugerament allunyat de la resta per sortir de l'aplicació. Per altra banda, s'ha implementat un menú de pausa que es mostra quan el jugador vol aturar el temps virtual del videojoc en qualsevol instant. Per a que les transicions entre escenes sigui més agradable, s'ha dissenyat una pantalla de càrrega que es mostra quan l'usuari selecciona una opció que requereix un canvi d'escena. En aquesta pantalla s'ha implementat una barra de càrrega que informa en tot moment al jugador del progrés de la càrrega.

7.3.1. Menu principal

Tenint en compte la importància de dissenyar un menú dinàmic i funcional s'ha optat per utilitzar com a fons el mateix escenari del mode un jugador. El dinamisme de l'escena s'aconsegueix amb un *script* que permetrà a la càmera rotar amb una direcció i una velocitat concreta. D'aquesta forma la imatge de fons sempre estarà en moviment i simularà que l'escena estigui "viva". Per millorar l'aspecte visual, s'han incorporat dues animacions al títol i a la resta d'elements del menú que donen sensació de fluïdesa i contribueixen en el dinamisme del conjunt.

S'han implementat quatre botons que realitzen funcions diferents. El botó "Un Jugador" l'enllaça amb l'escena "Level1" amb índex 1, que correspon al mode d'un jugador. El "Multijugador" activa l'escena amb índex 3 que correspon al mode multijugador.

Figura 7.11 *- Menu principa del videojoc.*

El botó "Opcions" desactiva el menú anterior i n'activa un de nou per accedir a l'opció de control de volum. Aquest menú secundari ha estat dissenyat per afegir noves opcions en una versió posterior del videojoc. Quan s'activa el menú d'opcions es permet tornar al principal prement el botó "Enrrere".

Figura 7.12 *- Menú d'opcions del videojoc*

7.3.2. Menu de pausa

En un joc d'aquest estil és imprescindible permetre a l'usuari pausa el transcurs del joc ja que es tracta d'una història llarga i que no s'interromp en cap moment. Per això s'ha dissenyat un menú de pausa que es mostra quan l'usuari prem la tecla "esc". Quan s'activa aquest menú, s'atura el temps virtual del joc i es reemprent quan l'usuari torna a prémer la mateixa tecla o bé prem el botó "Tornar". També es dona l'opció al jugador de sortir de la partida i tornar al menú principal amb el botó "Menú" o de sortir de l'aplicació amb el botó "Sortir".

Figura 7.13 - *Menu de pausa (in Game)*

El menú de pausa es gestiona amb un *script* propi anomenat *PauseMenu.* Aquest *script* conté les funcions *Pause(),* on s'escala el temps de joc a 0 (*Time.timeScale = 0f)* i la funció Resume(), que es torna a escalar a 1 *(Time.timeScale = 1f)*. També conté la funció *LoadMenu()* que amb el comandament *SceneManager.LoadScene()* accedeix novament al menú principal si el jugador prem l'opció corresponent. QuitGame() s'encarrega de sortir de l'aplicació amb el comandament *Application.Quit().*

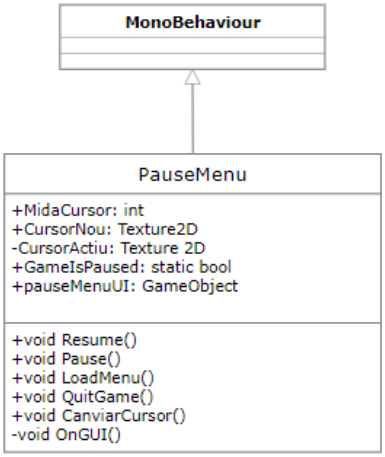


Figura 7.14 *- UML del Script PauseMenu.*

7.3.3. Pantalla de càrrega

Per millorar les transicions entre escenes se n'ha creat una que conté un *Canvas* amb una imatge negra de fons obtinguda del *Mercado de la Comunidad* de la plataforma *Steam*. També s'ha fet ús d'un *Slider* de color blau acotat entre 0 i 1 que mostra el procés de càrrega de l'escena juntament amb un *Text* que mostra numèricament l'estat d'aquest procés. Per donar més informació al jugador s'ha afegit un altre *GameObject* amb el component *Text* que assenyala que s'està realitzant aquest procés ("Carregant...").

S'ha fet ús de l'*script* explicat anteriorment *LevelLoader* per controlar l'aparició de l'escena de transició.

Figura 7.15 *- Pantalla de càrrega entre escenes*

7.3.4. Lobby

S'ha dissenyat una interfície per reunir als jugadors en una partida del mode multijugador. S'han implementat tres botons amb el mateix disseny que la resta de interfícies, per connectar-se al servidor i unir-se a una partida quan es donen les condicions establertes.

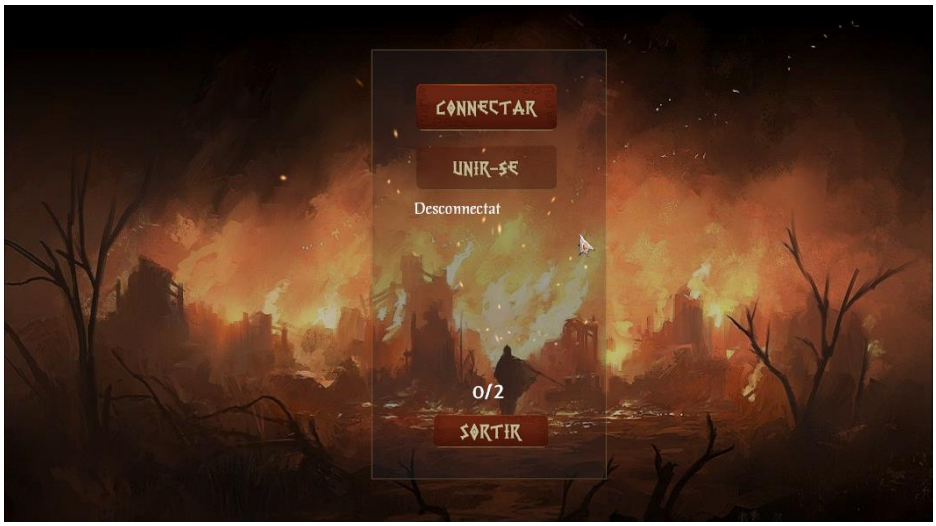


Figura 7.16 *- Pantalla de càrrega entre escenes*

Qui gestiona l'acció dels botons d'aquesta escena, així com la connexió amb el servidor per crear una partida és *l'script AutoLobby.*

Descripció de les variables

- **ConnectButton:** botó que connecta amb el servidor Photon Cloud.
- **JoinRandomButton:** botó que crea una partida o bé s'uneix a una existent.
- **playerCount:** variable que comptabilitza el número de jugadors connectats a la partida
- **Log:** text que dona informació dels esdeveniments a l'usuari.
- **PlayerCount:** text que mostra el número de jugadors connectats a la partida.
- **isLoading:** booleà que indica quan s'està canviant a l'escena de joc.
- **maxPlayersPerRoom:** indica un nombre de jugadors màxim d'una partida.
- **minPlayersPerRoom:** indica un nombre de jugadors mínim d'una partida.

Descripció dels mètodes principals

- **Connect():** mètode encarregat de connectar la sessió al servidor.
- **OnConnectedToMaster():** mètode que decideix l'activació dels botons en funció de l'estat de la partida.
- **OnJoinRandomFailed():** mètode que s'encarrega de mostrar un text quan s'ha creat una sala.
- **OnJoinedRoom():** mètode que mostra el text quan el jugador s'ha unit a una partida.
- **JoinRandom():** mètode que s'executa quan el jugador no s'ha pogut unir a una sala ja creada.
- **LoadMap():** encarregat de carregar l'escena multijugador quan hi ha un mínim de jugadors connectats al servidor.

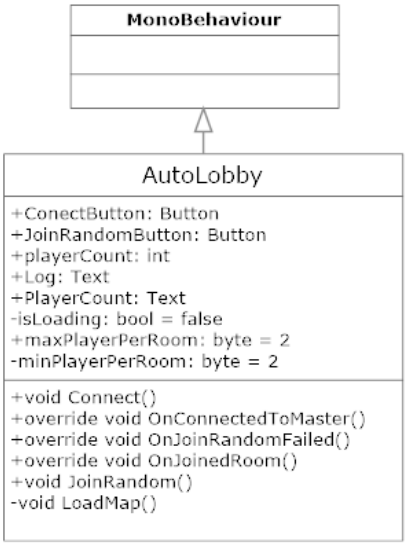


***Figura 7.17** - UML del Script AutoLobby*

7.3.5. Selecció de personatge

Quan els jugadors inicien una partida en mode multijugador apareix en primer lloc una pantalla de selección de personatge. Cada jugador escollirà un dels dos personatges per començar la partida. Per seleccionar el personatge, s'haurà de prémer un dels dos avatars que apareixen en pantalla i automàticament s'iniciarà la partida.

Figura 7.18 *- Pantalla de càrrega entre escenes*

L'*script RespawnController* és l'encarregat de l'aparició dels personatges quan l'usuari en selecciona un dels dos.

Descripció de les variables

- **respawnMale:** referència de la posició de sortida del jugador de sexe masculí.
- **respawnFemale:** referència de la posició de sortida del jugador de sexe femení.
- **malePrefab:** emmagatzema el *prefab* del jugador de sexe masculí.
- **femalePrefab:** emmagatzema el *prefab* del jugador de sexe femení.
- **canvas:** emmagatzema la interfície de selecció de personatge.

Descripció dels mètodes principals

- **RespawnPlayerMale():** mètode que instancia el prefab del jugador masculí en mode multijugador quan es selecciona el botó corresponent.
- **RespawnPlayerFemale():** mètode que instancia el prefab del jugador femení en mode multijugador quan es selecciona el botó corresponent.

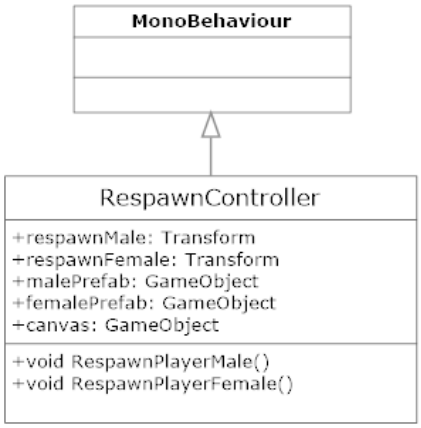


Figura 7.19 *- UML de l'script RespawnController.*

7.4. MockUp

Amb l'evolució de les interfícies gràfiques de l'usuari s'ha originat la necessitat de ampliar les formes de representació i millorar les existents per mostrar tota la informació que es vulgui transmetre.

Per al disseny de processos és important realitzar un diagrama que representi l'estructura, funcionament i comportament del sistema. Aquest diagrama s'ha de realitzar amb la informació que s'obté a mesura que es desenvolupa el projecte fins arribar a una proposta del producte final. Per tant, ha de mostrar un esbós del producte final, com s'organitzarà el seu contingut i el funcionament bàsic de tot el sistema.

Quan aquest diagrama s'utilitza per a la creació de videojocs, té l'objectiu de mostrar el contingut de les escenes, concretant els elements proposats en primera instància i ubicar-los en les pantalles del producte final.

MockUp, del inglès (esboç), és el diagrama que mostra una representació del disseny inicial amb poc detall dels elements d'un videojoc que s'utilitza durant el desenvolupament com a guia del procés de treball. Aquest model a escala serveix de demostració per assegurar la funcionalitat i la viabilitat del projecte, sent aquest susceptible a ser corregit i millorat en el disseny final. Compleix la funció de materialitzar un concepte i respon a la necessitat inicial de veure el producte integrat en un entorn real.

El *MockUp* dissenyat a l'inici del projecte consta de 4 pantalles relacionades entre si que es mostren depenent dels *inputs* que seleccioni l'usuari. Per confeccionar el disseny de les escenes, es va fer ús d'imatges del *asset* que s'ha utilitzat posteriorment per crear l'escenari del videojoc.

Figura 7.20 *- MockUp del disseny inicial.*

Al disseny posterior del primer esbós, s'han modificat les pantalles amb imatges reals del videojoc afegint escenes que prèviament no es contemplaven. Entre elles, la pantalla de càrrega com a transició entre escenes o el mode multijugador. Aquest mode de joc contempla la escena on es reuneixen els jugadors de la partida (*lobby*), la selecció de personatges i les pantalles que mostren la victòria del jugador. Totes les escenes finals retornen a l'escena principal i serà des d'allí on l'usuari podrà sortir de l'aplicació prement el botó corresponent. S'ha habilitat també l'opció de tancar l'aplicació mentre s'està desenvolupant la història del mode principal.

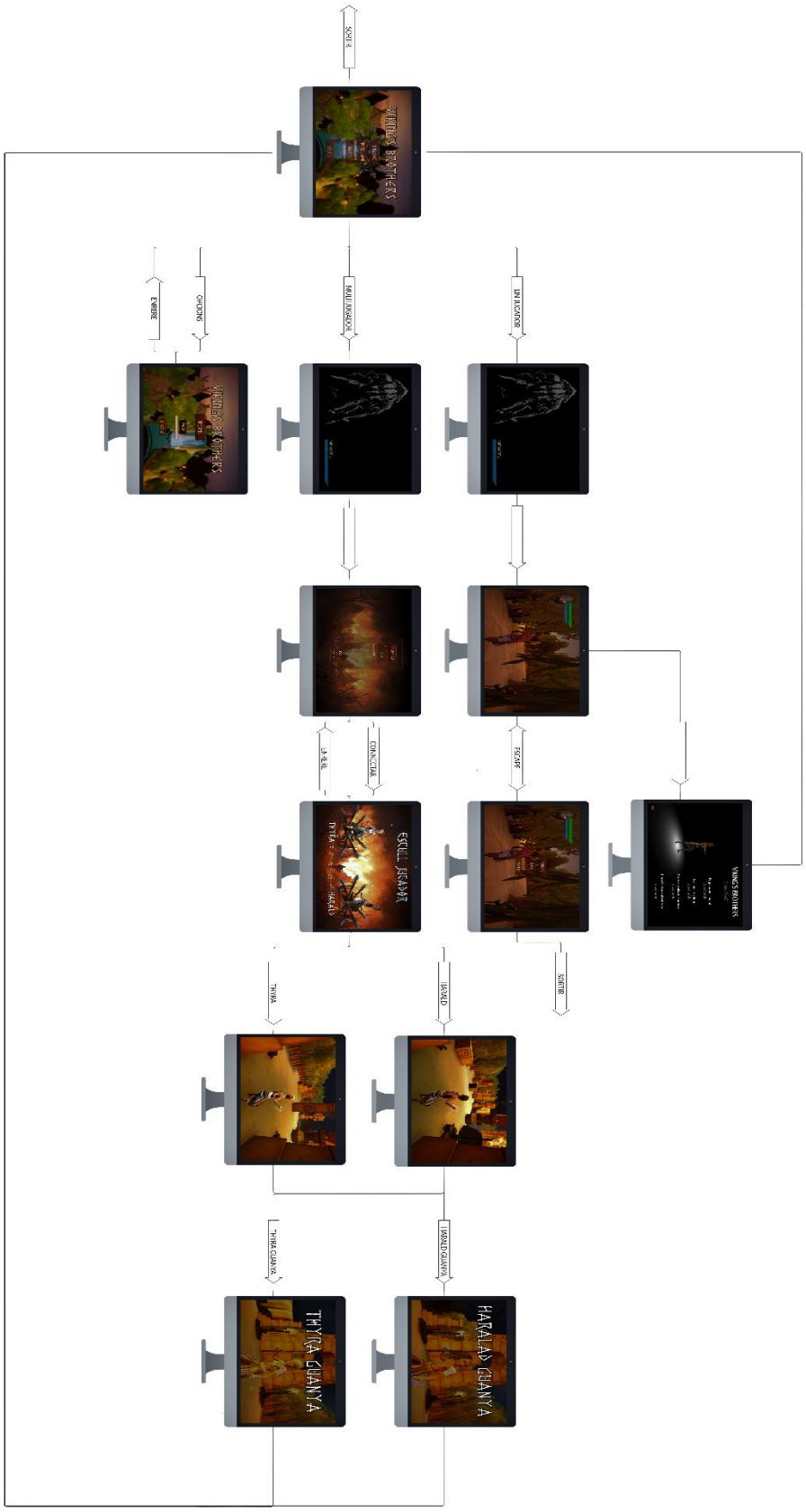


Figura 7.21 *- MockUp del disseny final.*

8. Photon Network 2.0

Photon Network és una plataforma de connexió per a videojocs en temps real. L'accés és gratuït fins un centenar de connexions simultànies. El preu varia en funció del nombre de persones connectades al servidor.

El software Photon Cloud permet realitzar proves de connectivitat amb facilitat sense preocupar-se dels servidors, ja que disposa dels seus servidors propis dedicats a les partides i aplicacions que estiguin connectats al seu servei.

Un servidor tracta d'un ordenador molt potent dedicat única i exclusivament a emmagatzemar informació online. En un videojoc, el servidor es dedica a allotja les partides, gestionar-les i sincronitzar-les per a tots els jugadors. Requereixen de personal que revisin que no hi hagi problemes de sobrecarrega, ni problemes de hardware.

La plataforma ofereix quatre tipus de connexió amb diferents característiques en funció dels requeriments del projecte on es vol implementar.

- **RealTime:** motor de xarxa multiplataforma per a videojocs de qualsevol genere de latència baixa.
- **PUN:** sistema RealTime dissenyat per ser implementat a Unity amb un gran rendiment.
- **Bolt:** solució basada amb esdeveniments pensat per a clients que connecti constantment el seus jugadors. S'utilitza en jocs d'estil FPS (*first person shooter*).
- **Quantum:** enfocat a xarxes deterministes. S'utilitza principalment per a jocs d'estil RTS (*real time strategy*).

En aquest projecte s'ha escollit el sistema de connexió PUN per desenvolupar el mode multijugador del videojoc ja que és el que presenta menys dificultats alhora d'implementar-lo.

8.1. Photon amb Unity 3D

Per implementar la plataforma Photon Network a Unity, es descarrega *l'asset* Photon 2 Unity Networking de l'Asset Store. Quan s'incorpora al projecte, apareix una finestra anomenada PUN Wizard que serveix per enllaçar el projecte amb la aplicació prèviament creada a la pàgina de Photon Cloud. Amb la identificació que et proporciona Photon quan es crea una aplicació, s'enllaça amb el projecte de Unity que vincula el vídeojoc amb l'aplicació creada.

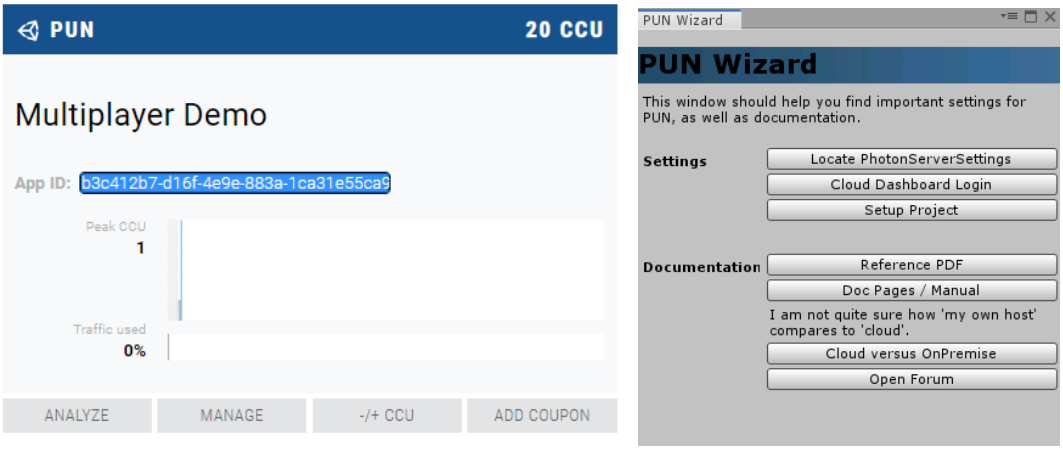


Figura 8.1 *- Aplicació PUN creada amb Photon.*

Un cop vinculat el projecte amb l'aplicació amb Photon Cloud, s'han de modificar els components dels objectes que es sincronitzaran durant la connexió online. Per una banda, cal modificar els *scripts* que gestionen el control de l'objecte.

S'ha de canviar la classe base d'on deriven els scripts implicats en la sincronització, si per defecte deriven de la classe *MonoBehaviour* s'hauran de modificar per a que derivin de *Photon.Pun.MonoBehaviourPun.*

Per sincronitzar la posició, la rotació i la execució de les animacions d'un personatge s'han d'afegir els següents components:

- **PhotonTransformView:** component necessari per sincronitzar la posició i rotació del personatge amb la resta de jugadors de la partida.

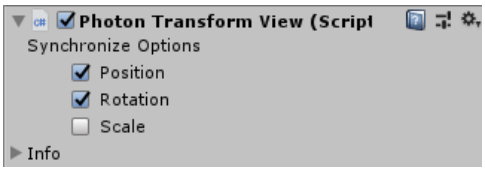


Figura 8.2 *- Component Photon Transform View del personatge*

- **PhotonAnimatorView:** component que s'utilitza per sincronitzar les animacions del personatge amb la resta de jugadors de la partida. Cada animació conté el seu paràmetre d'activació i aquest, s'ha de configurar segons sigui una variable continua o discreta. En el cas de les animacions dels personatges, tots els paràmetres es configuren com a variables continues.

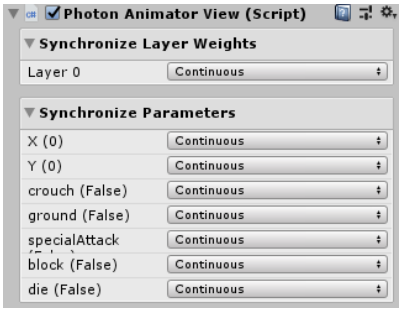


Figura 8.3 *- Component Photon Animator View del personatge*

Cal destacar que per a que les animacions es sincronitzen en tot moment es necessari modificar el paràmetre Culling Mode del component Animator i seleccionar l'opció de Always Animate.

- **PhotonView:** és el component que permetrà a la resta de jugadors visualitzar les variacions dels components *Transform* i *Animator.* Per seleccionar quins són els que es volen actualitzar, s'arrastra el component a la llista de *Observed Components* i es selecciona el tipus d'observació (*observe option*) en funció de la freqüència de actualització de la imatge:

  OFF: només s'utilitza per a enviar missatges RPC (qualsevol funció del avatar principal)
  UNRELIABLE: té una actualització continua, és a dir, s'envia informació constantment sense tenir en compte si s'estan realitzant canvis. Això provoca un consum més alt de recursos.
  UNRELIABLE ON CHANGE: nomes envia informació quan hi ha canvis de propietats. Quan les característiques es mantenen constants no s'intercanvia informació i per tant, es consumeixen menys recursos.
  RELIABLE DELTA COMPRESSED: és molt semblant a l'anterior amb la diferència que aquest tan sols envia informació de les propietats que han variat. El cas anterior envia tot el paquet de dades quan varia almenys una de les característiques.

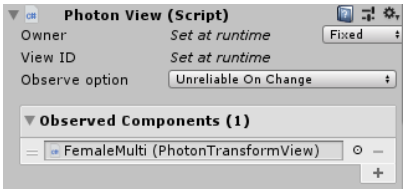


Figura 8.4. *- Component Photon View del personatge*

El funcionament del multijugador es basa en crear una rèplica del personatge principal a tots els dispositius connectats, és a dir, es formen avatars a partir de la versió original. S'ha de tenir clar que els avatars no tenen propietats sinó que són un reflex del model principal i per tant, serà ell l'encarregat de traslladar variacions en les seves propietats a la resta de rèpliques. Al model original se li aplicarà el comandament *photonView.IsMine* que detectarà quin és el jugador real de la partida, per distingir-lo de la còpia que es crea quan l'avatar s'instancia a les sessions dels jugadors interconnectats.

Photon té una forma particular d'instanciar objectes en mode multijugador, és a dir, objectes que estiguin sincronitzats entre tots els jugadors de la partida. Per invocar un prefab a l'escena, Photon obliga a accedir a una carpeta anomenada Resources per trobar el prefab que es vol instanciar. Per això, s'ha de canviar el nom de la carpeta que emmagatzema els prefabs del joc per a que pugui identificar el objecte correctament.

La manera que utilitza Photon per instanciar objectes a l'escena és diferent a fer-ho localment. Photon utilitza la funció *Photon.Pun.PhotonNetwork.Intantiate()* per instanciar prefabs en mode multijugador, és a dir, apareixerà una instància de l'objecte a tots els dispositius connectats a la partida.

```
Photon.Pun.PhotonNetwork.Instantiate(malePrefab.name, respawnMale.position, Quaternion.identity);
```

***Figura 8.5.**- Comandament per instanciar objectes en multijugador.*

9. Efectes visuals FX

El concepte FX fa referència a tots els efectes visuals que té un videojoc, és a dir, totes les animacions de elements que fan que el joc sigui més satisfactori i cridi més l'atenció de l'usuari.

Els efectes visuals són molt importants en un videojocs degut a que s'aconsegueix un acabat més professional, millora l'experiència del jugador i provoca que l'usuari tingui un *feedback* de les accions que està realitzant, incitant-lo a efectuar altres accions pel seu atractiu estètic. També ajuden a entendre el que està passant o a ressaltar llocs o objectes que interessa que el jugador centre l'atenció.

És important no abusar dels FX en un videojoc per no sobrecarregar l'escena i per no desviar l'atenció del jugador amb elements irrellevants. S'ha de trobar el terme mig per a que l'experiència sigui el més agradable i realista possible.

Es poden distingir dos tipus d'efectes visuals, les animacions de la interfície gràfica i els efectes que emfatitzen accions dins el joc.

9.1. Animacions de la interfície

Són els efectes que s'incorporen als elements de la UI per afegir dinamisme i transmetre a l'usuari que els components que permeten interaccions tenen vida i poden ser manipulats.

S'han incorporat animacions en la interfície del menú principal que modifiquen l'escalat del títol i la posició del menú.

Figura 9.1 *- Animacions del menú principal.*

9.2. Efectes d'accions dins el joc

S'incorporen efectes visuals en accions del jugador i en objectes de l'escenari per indicar que aquell element pot ser rellevant per al transcurs de la història.

Escut protector

S'ha implementat un efecte visual quan el jugador està blocant l'atac d'un enemic, és a dir, quan el jugador manté actiu l'estat "Block" dins la màquina d'estats. Consta d'una aura de color blau amb forma d'escut que gira al voltant del jugador mentre l'usuari manté la condició activa. *CharController* és l'*script* que controla l'activació d'aquest efecte.

Figura 9.2 *- FX de l'escut protector del jugador (in Game).*

Atac especial

Una de les habilitats del personatge principal és la capacitat de llençar una bola d'energia com a atac especial. També s'ha dissenyat un atac anàleg al del personatge principal per al mode multijugador. La diferència està amb el color i el so que emet quan aquest es llença sobre un objecte o enemic.

Figura 9.3 *- FX de l'escut protector del jugador (in Game).*

L'atac especial està format per una sèrie de components imprescindibles per a que sigui una arma funcional.

- **Transform:** component que determina la posició, rotació i escala de l'objecte.
- **ParticleSystem:** component que permet simular i renderitzar moltes imatges petites o malles, anomenades partícules, per produir un efecte visual. El sistema simula cada partícula col·lectivament per crear la impressió del efecte. Aquest component s'utilitza per crear objectes dinàmics com el foc, fum, líquids, etc. En aquest cas, s'han implementat dos sistemes de partícules provinents d'un *asset* que formen la bola de energía completa.
- **SphereCollider:** component necessari per detectar col·lisions amb altres objectes.
- **AudioSource**: aquest component s'encarrega de reproduir sons en un entorn 3D.
- **PhotonTransfromView:** component que sincronitza la posició, rotació i escala del *GameoObject* al mode multijugador amb Photon.
- **PhotonView:** aquest component es necessari per a la connexió multijugador amb Photon.

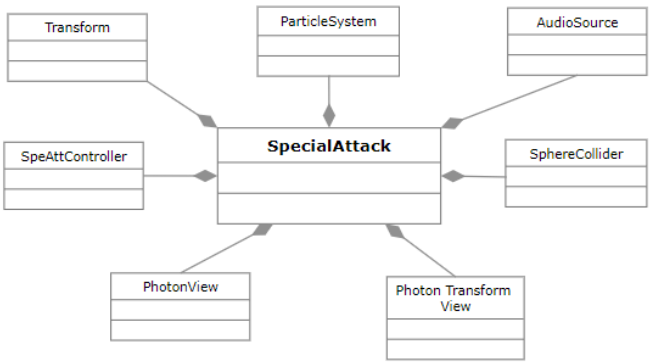


Figura 9.4 *- Components dels atacs especials.*

Per gestionar l'aparició i les característiques d'aquest FX, es necessari la implementació d'un *script* que ho controli. *SpeAttController* utilitza les següents variables per controlar l'atac especial del jugador.

Descripció de les variables

- **power:** indica la velocitat del projectil quan aquest es llençat pel jugador.
- **lifeTime:** indica el temps fins que l'objecte sigui destruït.
- **deltaTime:** variable que calcula el temps virtual
- **shootPoint:** referencia la posició de sortida de l'atac especial.
- **bulletPrefab:** emmagatzema el *prefab* de la bola d'energia que instanciarà el jugador.
- **explosionVFX:** emmagatzema el *prefab* que s'instancia quan la bola d'energia xoca amb un altre objecte o enemic.

Descripció dels mètodes principals

- **OnTriggerEnter():** mètode que detecta l'objecte de col·lisió i realitza accions en funció del seu *tag.*

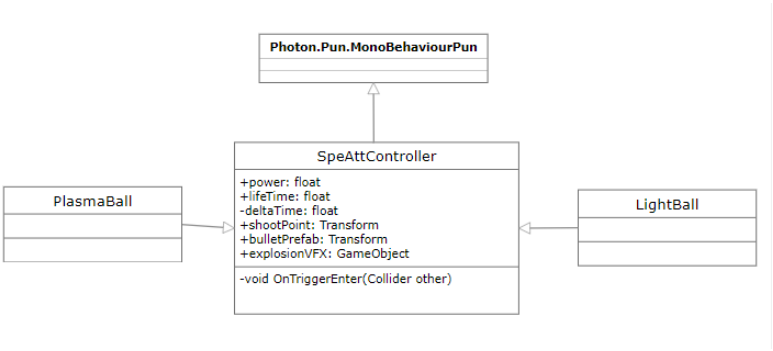


Figura 9.5 *- UML dels Scripts PlasmaController i LightController.*

Explosió de l'atac especial

Quan el jugador aconsegueix l'atac especial, apareix una explosió juntament amb l'activació d'una animació del personatge principal quan aquest interacciona amb l'objecte corresponent.

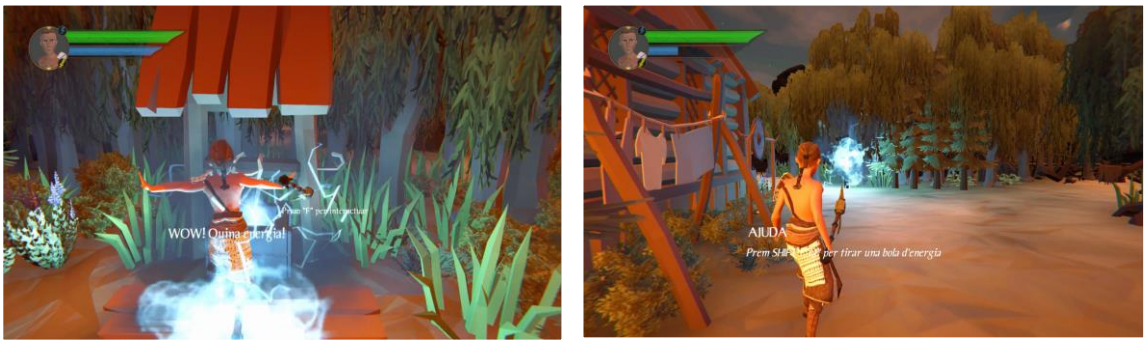


Figura 9.6 *- FX de l'explosió de l'atac especial (in Game).*

Aura al recuperar vida

Per emfatitzar quan el jugador recupera vida s'ha afegit un FX en forma de aura de color verd que envolta el jugador. S'ha implementat aquest efecte per indicar a l'usuari un increment de salut al interaccionar amb l'objecte corresponent. *PickUpObjects* és l'*script* que controla l'activació d'aquest efecte quan l'objecte que té al davant el jugador és un pou amb el foc de color verd.

Figura 9.7 *- FX de l'aura quan incrementa la salut del jugador (in Game).*

Llum de les claus

S'ha afegit una llum enfocant la clau per ressaltar l'objecte a l'escenari. S'ha configurat la llum del tipus "Spot" de manera que es concentra la intensitat de la llum en un punt concret. S'ha afegit un *script* anomenat *KeyController* que modifica la rotació de la llum en l'eix X per a que el jugador centre l'atenció en aquest punt.

Figura 9.8 *- FX de les claus (in Game).*

Moviment de la càmera

Quan hi ha una explosió a l'escenari s'ha afegit un efecte de moviment de la càmera que simula un atordiment del jugador causat per l'impacte de l'explosió. També emfatitza l'efecte incrementant la magnitud de l'explosió fent-la més visible per a l'usuari.

CameraShake és el controlador que s'utilitza per configurar l'efecte de moviment de la càmera. Es modifica els components de posició i escala i s'executa el mètode *Shake()* quan es produeix una explosió.

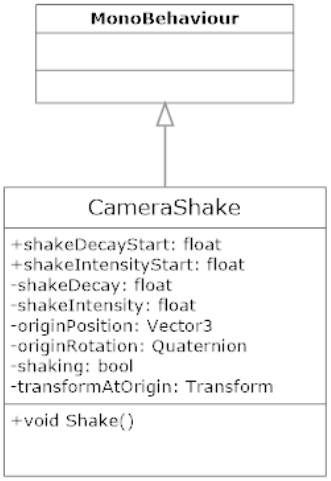


Figura 9.9 *- UML del Script CameraShake.*

Atac especial de l'enemic final

L'enemic final té una habilitat que no tenen la resta d'enemics. Tracta d'una bola d'energia amb un aspecte diferent a l'atac especial del jugador que llençarà quan aquest entri dins el radi d'acció de l'enemic.

Aquest element està format pels mateixos components que l'atac especial del jugador però es controlat amb un *script* diferent. *BlastController* gestiona l'aparició d'aquest element i la direcció que adopta quan l'enemic llança el seu atac.

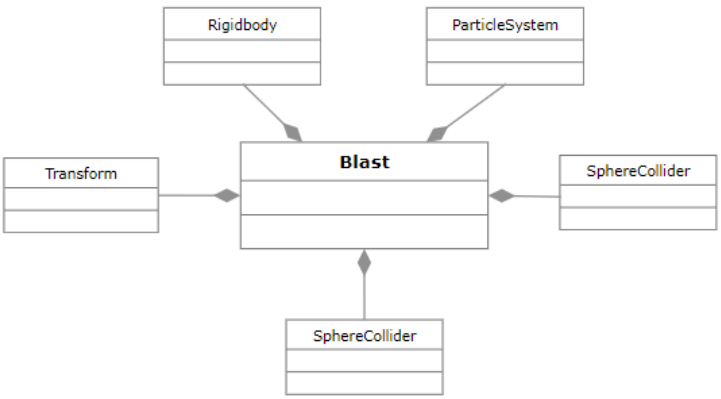


Figura 9.10 *- Components del atac especial del enemic final.*

Descripció de les variables

- **power:** indica la velocitat del projectil quan aquest es llençat per l'enemic.
- **lifeTime:** indica el temps fins que l'objecte sigui destruït.
- **deltaTime:** variable que calcula el temps virtual.
- **attackDamage:** indica la quantitat de dany de l'atac especial de l'enemic.
- **target:** és la referència de la posició del jugador que determina la direcció de sortida de l'atac.
- **explosionVFX:** emmagatzema el *prefab* que s'instancia quan la bola d'energia xoca amb el jugador.

Descripció dels mètodes principals

- **OnTriggerEnter():** mètode que s'encarrega de restar salut al jugador quan la bola d'energia impacta contra el *collider* del jugador.

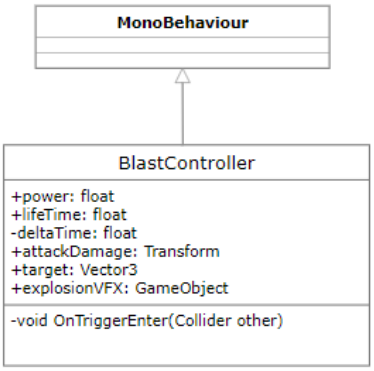


Figura 9.11 *- UML del Script BlastController.*

10. Música i efectes de so

El so d'un videojoc és un apartat tant important com els gràfics. Lo que genera la immersió en els videojocs és l'apartat sonor del mateix. Forma part d'un dels estímuls que rep el jugador i es capaç de transmetre emocions que afectaran a la jugabilitat del videojoc. Això provoca que l'usuari centri la seva atenció no tan sols en lo que veu, sinó en lo que escolta.

El so en un videojoc crea atmosferes, ens permet entendre històries, dóna vida a ambients i personatges. L'evolució amb el temps d'aquest aspecte ha donat lloc a ser un component immersiu de la història de gran importància. Per aquest motiu, el so en els videojocs es considera un art en si mateix.

En aquest projecte, s'han incorporat efectes de so que acompanyen a les accions dels personatges o dels enemics. També s'ha afegit un fil musical que dona ambient a la història i millora la jugabilitat. Tots els components de so estan gestionats des de diferents *scripts* segons qui l'estigui emetent. Per reproduir un so és imprescindible incorporar el component *Audio Source* per donar la capacitat a l'objecte de carregar un clip de so i reproduir-lo quan es donin les condicions programades.

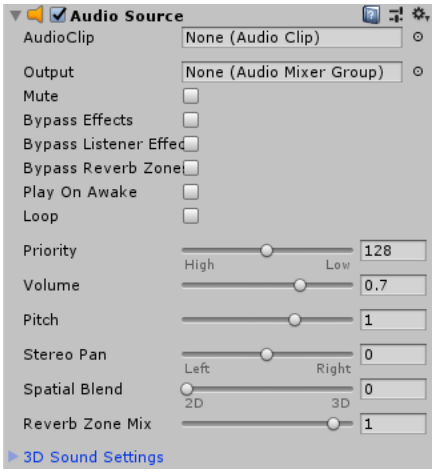


Figura 10.1 *- Component AudioSource.*

Hi ha sons de l'escena que es reprodueixen segons la distància entre l'emissor i el receptor. Existeix l'opció de configurar un so tridimensional per varia la intensitat del so en funció de la proximitat de l'usuari amb el so que s'està emetent. L'opció *Logarithmic Rollff* permet establir una distància mínima i màxima de emissió del so, de manera que si l'usuari es troba per sota de la distància mínima no sentirà res.

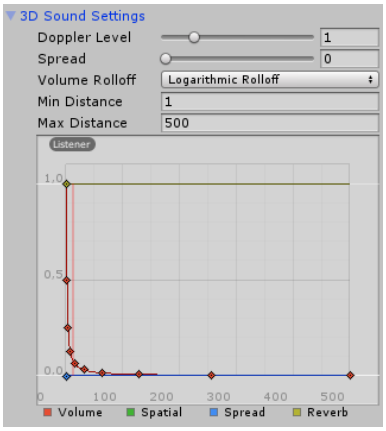


Figura 10.2 *- Comandament per carregar un clip de so al Audio Source..*

Per carregar un clip de d'àudio al reproductor, s'accedeix a la carpeta *Resources* on estan emmagatzemats els clips de àudio i es carrega al reproductor amb el comandament *Resources.Load<AudioClip>().* Seguidament, es reprodueix el clip de àudio amb el comandament *Play()*.

```
sceneAudio.clip = Resources.Load<AudioClip>("respawnEnemy");
sceneAudio.Play();
```

Figura 10.3 *- Comandament per carregar un clip de so al Audio Source.*

S'han implementat els següents sons gestionats per aquests *scripts*:

SCENE CONTROLLER

- **GetSpecial:** quan l'usuari aconsegueix col·locar un tresor a la plataforma corresponent.
- **RespawEnemy:** quan apareix un enemic al assolir una fita.
- **ExplosionFire:** so que acompanya l'explosió d'obertura d'un camí.

CHAR CONTROLLER

- **Attack:** quan el jugador executa l'atac principal.
- **Jump:** quan el jugador executa l'acció de saltar.

PICKUPOBJECTS

- **PickBox:** sona quan el jugador interacciona amb els tresors de l'escenari.
- **GetLife:** quan el jugador interacciona amb el pou per recuperar salut.
- **PickUp:** sona quan el jugador interacciona amb els cubs de l'escenari.
- **GetSpecialAttack:** quan el jugador aconsegueix l'atac especial.

- **OpenDoor:** so que s'executa quan es desbloqueja el primer portal de l'escenari.
- **BlockDoor:** sona quan es vol obrir el portal sense disposar de les claus.
- **YouWin:** so que es reprodueix quan el jugador interacciona amb l'objecte que desbloqueja el final.

PLAYERHEALTH

- **HitPlayer:** quan el jugador rep l'atac d'un enemic.
- **DiePlayer:** quan la salut del jugador disminueix fins a zero.

SPECIALCONTROLLER

- **SpecialAtt:** quan el jugador llença l'atac especial.

ENEMY CONTROLLER

- **HitEnemy:** quan l'enemic rep l'atac principal del jugador.
- **HitSpecialAttack:** quan l'enemic rep l'atac especial del jugador.
- **AttackPlayer:** so que s'executa quan el jugador es troba en rang d'atac de l'enemic.
- **PlayerFound:** so que s'executa quan el jugador entra dins el radi de visió de l'enemic.
- **DieEnemy:** quan l'enemic mort pels atacs del jugador.

La banda sonora de *Viking's Brothers* representa un ambient nòrdic i a la vegada hostil, afegint intriga i creant una atmosfera bèl·lica que dona credibilitat a la història que s'està narrant. La música conté efectes de so que representa la fauna de l'escenari i altres sons misteriosos que ajuden en la immersió de l'usuari dins el videojoc.

En aquest cas, s'ha afegit el clip de so directament al reproductor Audio Source de la càmera, amb l'opció *Play On Awake* activa per a que es reprodueixi tan bon punt s'iniciï la partida.

11. Optimització del projecte

A mesura que s'anaven incorporant elements al projecte, el rendiment gràfic es veia notablement afectat. Per a renderitzar qualsevol objecte a la pantalla, el CPU té que realitzar multitud de tasques que consumeixen molts recursos. Com més objectes visibles a l'escena, més treball tindrà la unitat CPU en renderitzar-los i mostrar-los en pantalla. Aquest fet va conduir a buscar fórmules per optimitzar el flux de treball i el contingut del projecte.

SUBSTITUCIÓ MESH COLLIDER A ALTRES COLLIDERS

El component *Mesh Collider* crea una malla que defineix perfectament la geometria del model 3D. La malla es creada a partir de milers de nodes que es combinen per formar l'estructura del objecte. És molt útil quan es requereix una definició perfecta de la seva arquitectura, però té les seves desavantatges ja que consumeix molts recursos gràfics. Per això s'ha substituït el component per altres configuracions de *colliders* als elements que no era imprescindible la definició exacta del model.

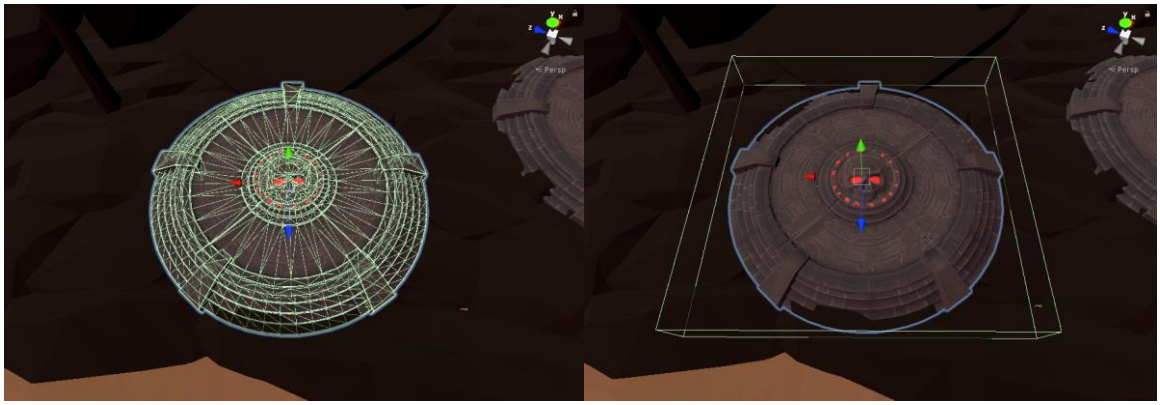


Figura 11.1 *- Substitució de Mesh Collider a Box Collider.*

DIMSINUIR QUALITAT DELS GRÀFICS

Una altra forma de reduir el consum de recursos del videojoc és disminuint la qualitat de renderitzat dels gràfics que es mostren a l'escena. Hi ha sis configuracions diferents que donen la possibilitat de modificar la resolució del renderitzat i la qualitat de les textures. S'ha escollit el nivell amb menor qualitat per millorar la fluïdesa del videojoc.

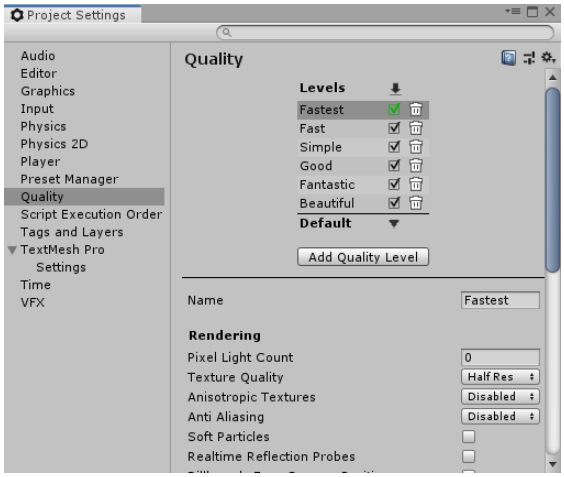


Figura 11.2 *- Qualitat del renderitzat.*

DISMINUIR LA DISTÀNCIA DE RENDERITZAT

Una de les opcions que ofereix el terreny on es disposen els elements de l'escena és el nivell de detall dels objectes des del punt de vista de la càmera. Es modifica el paràmetre *Base Map Distance* per disminuir la distància de renderitzat de l'escenari respecte la càmera del jugador. Pel que fa als arbres i objectes que conté l'escenari, s'ha disminuït el nivell de detall i la distància de visionat dels arbres i de la resta d'elements de l'escena.

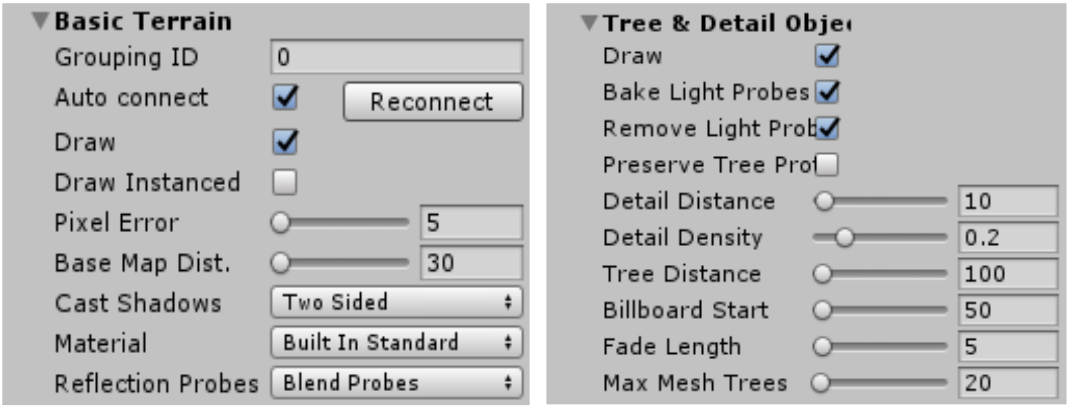


Figura 11.3 *- Paràmetres d'optimització del Terrain.*

12. Millores del projecte

En aquest apartat es detallaran les possibles millores que es podrien incorporar al videojoc. Aquestes mesures no s'han implantat perquè no es disposa dels coneixements necessàries per dur a terme la tasca, o bé per falta de temps. No obstant, algunes d'elles no s'han afegit per qüestions de prioritzar les tasques a realitzar. L'objectiu és implementar-les en un futur pròxim per satisfacció personal.

MILLORA DEL MENÚ D'OPCIONS

Actualment, el menú d'opcions tan sols disposa d'un control de volum de la música que s'està reproduint a l'escena. La voluntat és afegir diferents opcions de resolució de pantalla que s'adapti a les condicions del jugador i donar la possibilitat de canviar alguns dels controls del joc per a que l'usuari es senti més còmode.

AFEGIR MÉS D'UNA ARMA PRINCIPAL

A hores d'ara, l'escena està preparada per a que l'usuari agafi una atxa com a arma principal per derrotar els enemics que s'interposaran en el seu camí. L'addició de noves armes distribuïdes pel mapa milloraria la jugabilitat i l'experiència de l'usuari seria més positiva.

ELECCIÓ DEL PERSONATGE EN EL MODE UN JUGADOR

L'objectiu inicial era incorporar una elecció dels dos possibles personatges per desenvolupar la història que s'ha dissenyat. Aquesta tasca implica moltes variacions en la programació existent dels *scripts* i que, per tant, s'hauria d'invertir força temps en implementar-ho.

MILLORAR LES ANIMACIONS DEL PERSONATGE

Com s'ha explicat en capítols anteriors, el personatge executa les animacions en quatre direccions del moviment interconnectades en un arbre de mescla (*Blend Tree*). Per millorar la dinàmica del moviment, s'hauria d'implementar quatre animacions més a les posicions intermèdies entre dues direccions, és a dir, afegir animacions a l'arbre de mescla en les diagonals del pla XY.

Per millorar l'aspecte visual i afegir credibilitat a les accions de desenvolupa el jugador, s'hauria d'afegir una animació addicional per simular que el personatge té a les seves mans un objecte i l'està transportant d'un lloc a un altre. Aquesta mesura no s'ha implementat degut a la complexitat del moviment.

AFEGIR MÉS PUZLES A LA HISTÒRIA

A mesura que es progressava amb el disseny del videojoc, les idees dels puzles eren més abundants. El fet de tenir una data final per a l'entrega del projecte ha obligat a acotar les possibilitats i a centrar els esforços en altres aspectes més importants que la jugabilitat del videojoc.

MILLORAR LA JUGABILITAT DEL MODE MULTIJUGADOR

El mode multijugador tracta d'encadenar uns quants atacs al jugador contrari amb l'objectiu de derrotar-lo. Les mecàniques dels personatges són força simples i l'objectiu d'aquest mode de joc és limitat. Pròximament s'incorporaran objectes d'interacció, com els existents en el mode d'un jugador; que donaran habilitats als personatges i milloraran la jugabilitat del conjunt.

Conclusions

Al llarg d'aquest treball s'ha demostrat que un desenvolupador independent pot elaborar un projecte audiovisual a partir de les seves pròpies idees i capacitats, fent front a altres projectes desenvolupats per equips de treball amb més recursos. No es pretén mantenir una competència directa amb altres mètodes de treball sinó que són perfectament complementaris l'un amb l'altre. El resultat difícilment serà similar en quan al nivell de complexitat però existeix una relació directa entre les dues línies de treball, l'entreteniment dels usuaris.

El disseny modular ha permès estructurar el projecte de manera clara i ordenada. El seguiment d'una planificació inicial a donat lloc a la implementació d'un model de treball iteratiu i eficient. Mantenir aquest model com una acció sistemàtica al llarg del desenvolupament del videojoc ha resultat molt eficient degut a la constant recerca d'informació de totes les disciplines que engloba un projecte d'aquesta magnitud. No obstant, tot i millorar notablement el resultat final, ha obligat a modificar constantment el contingut incrementant la dedicació en cada punt i, per tant, el temps de realització de cada tasca.

La incorporació del mode multijugador ha estat un repte en si mateix. Ha obligat a emprendre un procés de recerca complex enfocat a un model de connexió *online* utilitzant un servidor proporcionat per una plataforma de sincronització a temps real. Els objectius d'afegir aquesta modalitat de joc es van centrar amb la investigació de com implementar-ho al sistema. Per això, el mode multijugador té carències en quan al disseny de l'escenari i funcionalitats dels jugadors. L'objectiu principal es va assolir quan dos jugadors connectats amb una xarxa local podien interaccionar entre ells i el comportament de l'escena era el mateix en tots els dispositius. Aquest ha estat un dels objectius que s'han aconseguit al llarg del projecte, però cal destacar-ne els següents:

- Estructurar un projecte d'alta complexitat a partir d'un disseny modular.
- Projectar un model 3D i tenir la capacitat d'implementar-lo en un software.
- Domini del motor de videojocs utilitzat amb un nivell alt.
- Assolir la capacitat de programar amb el llenguatge C Sharp.
- Entendre el funcionament dels servidors i del sistema de connectivitat online.
- Coneixement de la plataforma Photon Networking i de les seves funcions.
- Crear i interpretar diagrames de classe i UML.
- Exportar un projecte per a qualsevol plataforma compatible amb el *game engine.*

En definitiva, l'objectiu de dissenyar un videojoc d'aquestes característiques no girava entorn al resultat final sinó al camí per arribar fins allí. A mesura que avançava amb el projecte, anava despertant noves idees donant lloc a la incertesa del producte final, enfocant la idea del desenvolupament en "fins on puc arribar". La transversalitat d'aquesta disciplina ha conduït a emprar totes les meves capacitats per fer realitat aquest projecte, exercint totes les qualitats obtingudes al llarg de la carrera. La complexitat d'un videojoc obliga a explotar totes les habilitats i competències d'un mateix, característiques pròpies d'un enginyer.

Pressupost

Es realitzarà una estimació del cost de producció i disseny d'aquest videojoc en funció de les tasques que hauria desenvolupat cada perfil professional. S'assignarà un sou a cada membre de l'equip segons el seu paper dins el projecte. S'establirà una quantitat econòmica de referència per jornada de treball.

Es contemplaran els següents perfils professionals per desenvolupar un videojoc d'aquestes característiques:

- Departament de disseny (dissenyadors, guionistes, nivells)
- Departament gràfic i artístic (dibuixants, modeladors, animadors)
- Departament de so (música, efectes de so)
- Departament de programació (programadors, IA)
- Departament d'anàlisi de software (disseny d'scripts, UML)
- Departament de control de qualitat (provadors del joc)

El número de jornades corresponents a cada perfil professional es veuen reflectides en la taula del annex A2.

Perfil professional	**Cost per jornada [€]**	**Nº de jornades**	**Cost [€]**
Disseny (DIS)	200	104	20.800
Gràfics (GRA)	200	46	9.200
Sons (SO)	190	15	2.850
Programació (PRO)	250	142	35.500
Software (SOF)	230	50	11.500
Control de qualitat (QUA)	180	30	5.400
TOTAL			**85.250 €**

Taula P.1 *– Cost de les tasques del projecte*

S'ha tingut en compte el cost computacional que deriva del desgast dels equips electrònics que s'han utilitzat durant el procés.

Equip	**Preu [€]**	**Vida útil**	**Temps d'utilització**	**Cost [€]**
PC ASUS K550C	650	4 anys	0,75 anys	122
Monitor T200HD	210	6 anys	0,75 anys	27
TOTAL				**149 €**

Taula P.2 *– Cost computacional del projecte*

El **cost total** que ha representat el disseny i desenvolupament d'aquest videojoc són **85.400 €**

Bibliografia

[1] LIDON MAÑAS, Marc. *Unity 3D.* 1ª ed. Barcelona: S.A. MARCOMBO, 2018.

[2] VAN DER HEYDE, Fien. *UML 2.5: DOMINE EL DISEÑO CON LOS PATRONES DE DISEÑO.* 2ª ed. Barcelona: ENI, 2019.

[3] LAURENT DEBRAUWER, FIEN VAN DER HEYDE. *UML 2 MODELIZACIÓN DE OBJETOS.* 2ª ed. Barcelona: ENI, 2010.

[4] ASTERIOS AGKATHIDIS, *MODULAR STRUCTURES IN DESIGN AND ARCHITECTURE.* 1ª ed. Amsterdam: NL. DESIGN, 2009.

[5] CRISTOPHER RUSH, *Programming the Photon: Getting Started with the Internet of Things*. 1ª ed. New York: McGraw-Hill Education, 2016.

[6] FCO. JAVIER CEBALLOS SIERRA, *Microsoft C#. Lenguaje y Aplicaciones.* 2ª ed. Madrid: RA-MA, 2007.

[7] ARJAN EGGES, JEROEN D. FOKKER, MARK H. OVERMARS, *Learning C# by Programming Games.* 1ª ed. Berlin: Springer Verlag Berlin and Heidelberg GmbH Co KG, 2013.

Webgrafia

[1] Unity User Manual *(última visita: 02 de jun. 2020)*

https://docs.unity3d.com/Manual/index.html

[2] The Visual Workspace For Team Collaboration *(última visita: 16 de abr. 2020)*

https://app.creately.com/diagram/SB8N9wkRcjz/edit

[3] Easy and Powerful *(última visita: 26 de may. 2020)*

https://cloud.smartdraw.com/

[4] Free Sound Effects *(última visita: 04 de mar. 2020)*

https://www.freesoundeffects.com/

[5] Get animated, Animate 3D characters for games, film, and more. *(última visita: 08 de abr. 2020)*

https://www.mixamo.com/#/?page=1&type=Motion%2CMotionPack

[6] Unity for spanish users *(última visita: 06 de feb. 2020)*

https://forum.unity.com/threads/unity-for-spanish-users.14987/

[7] We Make Multiplayer Simple *(última visita: 26 de may. 2020)*

https://www.photonengine.com/en/Photon

[8] Unified Modeling Language (UML) | Class Diagrams *(última visita: 02 de may. 2020)*

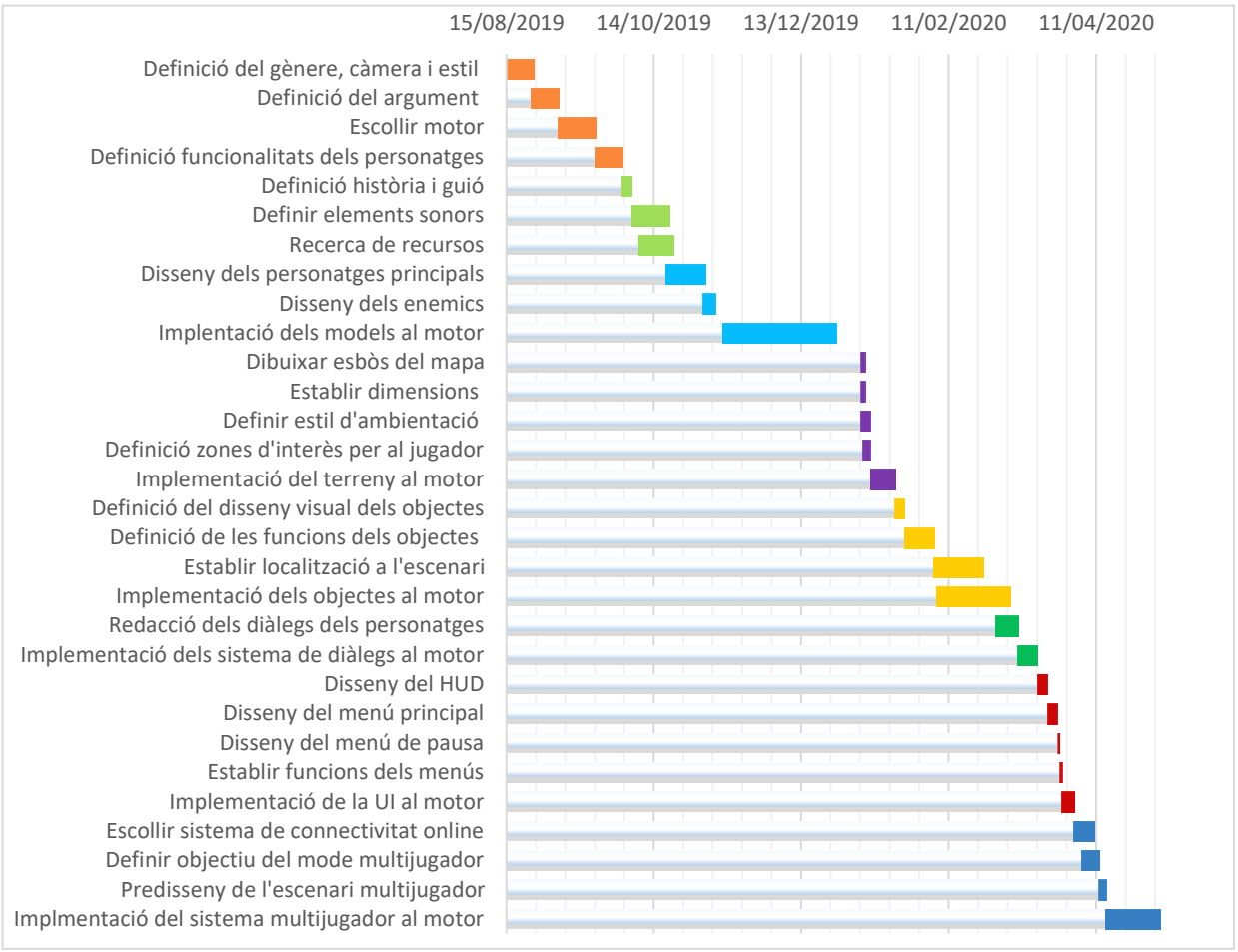
https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/?ref=rp

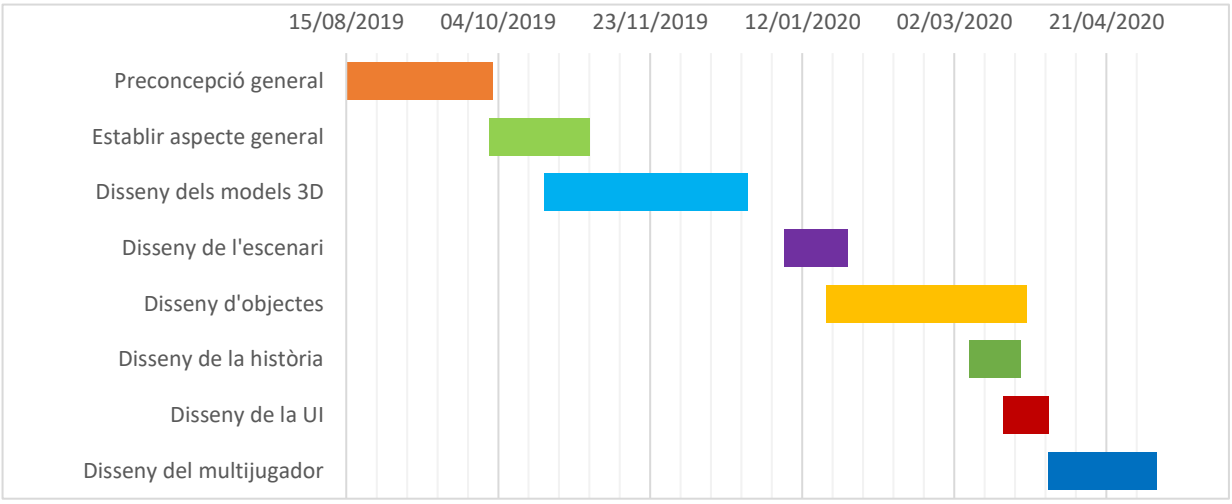
[9] Iterative Model: What Is It And When Should You Use It? *(última visita: 05 de jun. 2020)*

https://airbrake.io/blog/sdlc/iterative-model#:~:text=The%20iterative%20model%20is%20a,the%20final%20system%20is%20complete.

Annex A

A1. Diagrama de Gantt





A2. Distribució de tasques

DIS	1.1	Definició del gènere, càmera i estil	15/08/2019	26/08/2019	11
DIS	1.2	Definició del argument	25/08/2019	05/09/2019	11
DIS	1.3	Escollir motor	05/09/2019	20/09/2019	15
DIS	1.4	Definició funcionalitats dels personatges	20/09/2019	01/10/2019	11
DIS	2.1	Definició història i guió	01/10/2019	05/10/2019	4
SO	2.2	Definir elements sonors	05/10/2019	20/10/2019	15
GRA	2.3	Recerca de recursos	08/10/2019	22/10/2019	14
GRA	3.1	Disseny dels personatges principals	19/10/2019	04/11/2019	16
GRA	3.2	Disseny dels enemics	03/11/2019	08/11/2019	5
PRO	3.3	Implentació dels models al motor	11/11/2019	27/12/2019	46
GRA	4.1	Dibuixar esbòs del mapa	06/01/2020	08/01/2020	2
GRA	4.2	Establir dimensions	06/01/2020	08/01/2020	2
DIS	4.3	Definir estil d'ambientació	06/01/2020	10/01/2020	4
DIS	4.4	Definició zones d'interès per al jugador	07/01/2020	10/01/2020	3
PRO	4.5	Implementació del terreny al motor	10/01/2020	20/01/2020	10
GRA	5.1	Definició del disseny visual dels objectes	20/01/2020	24/01/2020	4
PRO	5.2	Definició de les funcions dels objectes	24/01/2020	05/02/2020	12
DIS	5.3	Establir localització a l'escenari	05/02/2020	25/02/2020	20
PRO	5.4	Implementació dels objectes al motor	06/02/2020	07/03/2020	30
DIS	6.1	Redacció dels diàlegs dels personatges	01/03/2020	10/03/2020	9
PRO	6.2	Implementació dels sistema de diàlegs al motor	10/03/2020	18/03/2020	8
DIS	7.1	Disseny del HUD	18/03/2020	22/03/2020	4
DIS	7.2	Disseny del menú principal	22/03/2020	26/03/2020	4
DIS	7.3	Disseny del menú de pausa	26/03/2020	27/03/2020	1
PRO	7.4	Establir funcions dels menús	27/03/2020	28/03/2020	1

PRO	7.5	Implementació de la UI al motor	28/03/2020	02/04/2020	5
PRO	8.1	Escollir sistema de connectivitat online	02/04/2020	10/04/2020	8
DIS	8.2	Definir objectiu del mode multijugador	05/04/2020	12/04/2020	7
GRA	8.3	Predisseny de l'escenari multijugador	12/04/2020	15/04/2020	3
PRO	8.4	Implmentació del sistema multijugador al motor	15/04/2020	07/05/2020	22

A3. Codis font

...r Game\MultiplayerGame\Assets\Scripts\CharController.cs 1

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine.AI;
5  using UnityEngine;
6
7  namespace controller
8  {
9      public class CharController : Photon.Pun.MonoBehaviourPun
10     {
11         GameObject hud;
12         ManaController manaController;
13
14         PlayerHealth playerHealth;
15         PickUpObjects pickUpObjects;
16         GameObject crouchTrunk1;
17         GameObject crouchTrunk2;
18         GameObject shield;
19         GameObject lifeShield;
20
21         Transform tr;
22         Rigidbody rg;
23         Animator anim;
24         AudioSource audioPlayer;
25         AudioSource audioShield;
26
27         public CapsuleCollider coll;
28         public CapsuleCollider axeColl;
29
30         public GameObject smc;
31         public GameObject weapon;
32
33         public Transform CameraShoulder;
34         public Transform CameraHolder;
35         private Transform cam;
36
37         private float rotY = 0f;
38
39
40         public CharStats Stats;
41
42         public bool Active = true;
43         public bool Player = true;
44
45         public bool OnGround = false;
46         public bool Jumping = false;
47         public bool Crouch = false;
48         public bool Crouching = false;
49         public bool Running = false;
50         public bool Attacking = false;
51         public bool AxeMotion = false;
52         public bool specialAttacking = false;
53         public bool Block = false;
54         public bool Interact = false;
55         public bool Interacting = false;
56
```

...r Game\MultiplayerGame\Assets\Scripts\CharController.cs 2

```
57          private Vector2 moveAnim;
58          private Vector2 moveDelta;
59          private Vector2 mouseDelta;
60
61          private float speed;
62          private float deltaT;
63          private float damage;
64          public bool inWater = false;
65          private bool repeatSound;
66
67          public InputController _input;
68
69
70          // Start is called before the first frame update
71          void Start()
72          {
73              hud = GameObject.Find("HUDCanvas");
74              manaController = hud.GetComponent<ManaController>();
75
76              crouchTrunk1 = GameObject.Find("CrouchTrunk1");
77              crouchTrunk2 = GameObject.Find("CrouchTrunk2");
78              shield = GameObject.Find("Shield");
79              lifeShield = GameObject.Find("LifeShield");
80
81              Cursor.visible = false;
82              smc.SetActive(false);
83              tr = this.transform;
84
85              pickUpObjects = GetComponent<PickUpObjects>();
86              playerHealth = GetComponent<PlayerHealth>();
87              rg = GetComponent<Rigidbody>();
88              anim = GetComponent<Animator>();
89              audioPlayer = GetComponentInParent<AudioSource>();
90              audioShield = GetComponentInChildren<AudioSource>();
91
92              axeColl.enabled = false;
93              weapon.SetActive(false);
94
95              cam = Camera.main.transform;
96          }
97
98          void FixedUpdate()
99          {
100             if (Player)
101             {
102                 PlayerControl();
103                 CameraControl();
104             }
105             else
106             {
107                 Player = photonView.IsMine;
108             }
109             if (!Active)
110             {
111                 return;
112             }
```

...r Game\MultiplayerGame\Assets\Scripts\CharController.cs 3

```
113
114             MoveControl();
115             AnimControl();
116         }
117
118         private void PlayerControl()
119         {
120             _input.Update();
121
122             float deltaX = _input.CheckF("Horizontal");
123             float deltaZ = _input.CheckF("Vertical");
124             float mouseX = _input.CheckF("Mouse X");
125             float mouseY = _input.CheckF("Mouse Y");
126             Jumping = _input.Check("Jump");
127             Crouch = _input.Check("Crouch");
128             AxeMotion = _input.Check("Axe");
129             Attacking = _input.Check("Attack");
130             specialAttacking = _input.Check("SpecialAttack");
131             Block = _input.Check("Block");
132             Interacting = _input.Check("Interact");
133
134
135             moveDelta = new Vector2(deltaX, deltaZ);
136             mouseDelta = new Vector2(mouseX, mouseY);
137
138             deltaT = Time.deltaTime;
139
140
141         }
142         private void MoveControl()
143         {
144
145             Vector3 side = Stats.speed * moveDelta.x * deltaT * tr.right;
146             Vector3 forward = Stats.speed * moveDelta.y * deltaT * tr.forward;
147             Vector3 endSpeed = side + forward;
148
149
150             RaycastHit hit;
151             OnGround = Physics.Raycast(this.tr.position, -tr.up, out hit,
                  0.4f);
152
153             if (OnGround)
154             {
155                 if (specialAttacking | Attacking | Block | Interacting |
                      playerHealth.playerDead)
156                     rg.constraints = RigidbodyConstraints.FreezeAll;
157                 else
158                     rg.constraints = RigidbodyConstraints.FreezeRotation;
159
160                 if (Block && AxeMotion && pickUpObjects.playerWeapon)
161                 {
162                     audioShield.enabled = true;
163                     shield.SetActive(true);
164                 }
165                 else
166                 {
```

...r Game\MultiplayerGame\Assets\Scripts\CharController.cs 4

```
167                     audioShield.enabled = false;
168                     shield.SetActive(false);
169                 }
170 
171                 if (pickUpObjects.activeShield)
172                     lifeShield.SetActive(true);
173                 else
174                     lifeShield.SetActive(false);
175 
176 
177                 if (Attacking && pickUpObjects.playerWeapon && AxeMotion)
178                 {
179                     audioPlayer.clip = Resources.Load<AudioClip>("attack");
180                     audioPlayer.Play();
181 
182                     axeColl.enabled = true;
183                 }
184                 else
185                     axeColl.enabled = false;
186 
187                 if (Crouch)
188                 {
189                     crouchTrunk1.GetComponent<MeshCollider>().enabled = false;
190                     crouchTrunk2.GetComponent<MeshCollider>().enabled = false;
191                 }
192                 else
193                 {
194                     crouchTrunk1.GetComponent<MeshCollider>().enabled = true;
195                     crouchTrunk2.GetComponent<MeshCollider>().enabled = true;
196                 }
197 
198                 if (Crouching)
199                     OnCrouch();
200                 else
201                 {
202                     if (AxeMotion)
203                     {
204                         endSpeed *= Stats.AxeSpeedDecrement;
205                         smc.SetActive(true);
206 
207                     }
208                     else
209                         smc.SetActive(false);
210                 }
211 
212                 if (Jumping)
213                 {
214                     if (Crouch)
215                         OnCrouch();
216 
217                     else
218                         Jump();
219                 }
220             }
221             else
222             {
```

...r Game\MultiplayerGame\Assets\Scripts\CharController.cs 5

```
223                 if (Crouch)
224                 {
225                     OnCrouch();
226                 }
227 
228             }
229 
230             Vector3 sp = rg.velocity;
231             endSpeed.y = sp.y;
232 
233             rg.velocity = endSpeed;
234 
235             moveAnim = moveDelta * (AxeMotion ? 1 : 1);
236         }
237 
238         public void Jump()
239         {
240             audioPlayer.clip = Resources.Load<AudioClip>("jump");
241             audioPlayer.Play();
242 
243             rg.AddForce(tr.up * Stats.jumpForce);
244         }
245 
246         public void OnCrouch()
247         {
248             Crouch = !Crouch;
249             Crouching = false;
250 
251         }
252 
253         public void OnTriggerEnter(Collider other)
254         {
255             if (other.gameObject.tag == "Interactuable")
256             {
257                 Interact = true;
258             }
259             if (other.gameObject.tag == "Water")
260             {
261                 inWater = true;
262                 playerHealth.Death();
263             }
264             if(other.gameObject.tag == "Fire")
265             {
266                 playerHealth.Death();
267             }
268         }
269 
270         public void OnTriggerExit(Collider other)
271         {
272             if (other.gameObject.tag == "Interactuable")
273             {
274                 Interact = false;
275             }
276         }
277 
278         private void CameraControl()
```

...r Game\MultiplayerGame\Assets\Scripts\CharController.cs 6

```
279         {
280             rotY += mouseDelta.y * deltaT * Stats.rotationSpeed;
281             float xrot = mouseDelta.x * deltaT * Stats.rotationSpeed;
282 
283             tr.Rotate(0, xrot, 0);
284 
285             rotY = Mathf.Clamp(rotY, Stats.minAngle, Stats.maxAngle);
286 
287             Quaternion localRotation = Quaternion.Euler(-rotY, 0, 0);
288             CameraShoulder.localRotation = localRotation;
289 
290             cam.position = Vector3.Lerp(cam.position, CameraHolder.position,
                Stats.cameraSpeed * deltaT);
291             cam.rotation = Quaternion.Lerp(cam.rotation,
                CameraHolder.rotation, Stats.cameraSpeed * deltaT);
292         }
293 
294 
295         private void AnimControl()
296         {
297             anim.SetBool("crouch", Crouch);
298             anim.SetBool("ground", OnGround);
299             anim.SetBool("axe", AxeMotion);
300             anim.SetFloat("X", moveAnim.x);
301             anim.SetFloat("Y", moveAnim.y);
302 
303             if (AxeMotion && pickUpObjects.PickedObject == null)
304             {
305                 anim.SetBool("attack", Attacking);
306                 anim.SetBool("block", Block);
307                 anim.SetBool("specialAttack", specialAttacking);
308             }
309 
310             if (Interact)
311               anim.SetBool("interact", Interacting);
312 
313             if (pickUpObjects.playerWeapon)
314                 anim.SetBool("playerWeapon", true);
315 
316             if (!Crouch && !AxeMotion && !Attacking && !specialAttacking && !
                Block && OnGround && !Interact && !playerHealth.playerDead)
317                 anim.Play("Walk");
318         }
319     }
320 
321 }
322 
```

... Game\MultiplayerGame\Assets\Scripts\EnemyController.cs 1

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5  using UnityEngine.AI;
6
7  namespace controller
8  {
9
10     public class EnemyController : MonoBehaviour
11     {
12         GameObject sceneController;
13         GameObject player;
14         PlayerHealth playerHealth;
15         AudioSource audioEnemy;
16         AudioSource audioPlayer;
17         AudioSource audioScene;
18
19         Transform playerPosition;
20
21         public GameObject blood;
22
23         Rigidbody rg;
24         Animator anim;
25
26         Vector3 inicialPosition;
27         NavMeshAgent nav;
28         CapsuleCollider coll;
29
30         public bool playerinRange;
31         public bool dieEnemy;
32         public bool walktoPlayer = false;
33         public bool attackPlayer = false;
34         private bool repeatSound1 = true;
35         private bool repeatSound2 = true;
36         private bool repeatSound3 = true;
37
38         private float timer;
39         public float timeBetweenAttacks = 0.5f;
40         public int attackDamage = 10;
41
42         public float visionRadius;
43         public float speed;
44         public float damageAxe;
45         public float damageSpecialAtt;
46         public float health;
47
48         void Start()
49         {
50             sceneController = GameObject.Find("SceneController");
51             player = GameObject.FindGameObjectWithTag("Player");
52             playerHealth = player.GetComponent<PlayerHealth>();
53             audioPlayer = player.GetComponentInParent<AudioSource>();
54             audioScene = sceneController.GetComponent<AudioSource>();
55             playerPosition = GameObject.FindGameObjectWithTag
                 ("Player").transform;
```

... Game\MultiplayerGame\Assets\Scripts\EnemyController.cs 2

```
56
57              audioEnemy = GetComponent<AudioSource>();
58
59              anim = GetComponent<Animator>();
60              rg = GetComponent<Rigidbody>();
61
62              nav = GetComponent<NavMeshAgent>();
63              inicialPosition = transform.position;
64          }
65
66          void FixedUpdate()
67          {
68              AnimControl();
69              Die();
70              FollowPlayer();
71              SoundControl();
72          }
73
74          public void Update()
75          {
76              timer += Time.deltaTime;
77
78              if(timer >= timeBetweenAttacks && playerinRange)
79              {
80                  Attack();
81              }
82              if (playerHealth.currentHealth <= 0)
83              {
84                  walktoPlayer = false;
85              }
86
87          }
88
89          public void Attack()
90          {
91              timer = 0f;
92
93              if(playerHealth.currentHealth > 0 && !dieEnemy)
94              {
95                  playerHealth.TakeDamage(attackDamage);
96              }
97
98          }
99
100         public void OnTriggerEnter(Collider other)
101         {
102             if (health > 0)
103             {
104
105                 if (other.gameObject.tag == "Axe")
106                 {
107                     audioEnemy.clip = Resources.Load<AudioClip>("hitEnemy");
108                     audioEnemy.Play();
109
110                     Vector3 posBlood = transform.position + (new Vector3(0,
                        1.7f, 0));
```

... Game\MultiplayerGame\Assets\Scripts\EnemyController.cs 3

```
111                     GameObject bloodVfx = Instantiate(blood, posBlood,
                        Quaternion.identity);
112                     Destroy(bloodVfx, 2);
113
114                     health = health - damageAxe;
115                 }
116                 if (other.gameObject.tag == "SpecialAttack")
117                 {
118
119                      audioEnemy.clip = Resources.Load<AudioClip>
                        ("hitSpecialAttack");
120                      audioEnemy.Play();
121
122                      health = health - damageSpecialAtt;
123
124                 }
125
126
127                 if (other.gameObject.tag == "Player")
128                 {
129                     playerinRange = true;
130                     attackPlayer = true;
131                 }
132             }
133
134         }
135         public void OnTriggerExit(Collider other)
136         {
137             if (other.gameObject.tag == "Player")
138             {
139                 playerinRange = false;
140                 attackPlayer = false;
141
142             }
143
144         }
145
146         public void FollowPlayer()
147         {
148             float dist = Vector3.Distance(playerPosition.transform.position,
                  transform.position);
149
150             if (dist < visionRadius)
151             {
152                 walktoPlayer = true;
153                 nav.SetDestination(playerPosition.transform.position);
154
155             }
156             else
157                 walktoPlayer = false;
158         }
159         private void SoundControl()
160         {
161             if (attackPlayer && repeatSound1)
162             {
163                 audioEnemy.clip = Resources.Load<AudioClip>("attackPlayer");
```

... Game\MultiplayerGame\Assets\Scripts\SceneController.cs 1

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using TMPro;
6  using UnityEngine.SceneManagement;
7
8
9  namespace controller
10 {
11     public class SceneController : MonoBehaviour
12     {
13         public GameObject youWin;
14         public GameObject gameOver;
15
16         PickUpObjects keysPlayer;
17         GameObject loadScene;
18         LevelLoader levelLoader;
19
20         private float timerDead;
21         private float timer;
22         private float timeToEnd;
23
24         public GameObject explosionVFX;
25         public Transform explosionPosition;
26
27         GameObject cubeGreen;
28         GameObject cubeBlue;
29         GameObject cubeRed;
30         GameObject portalFinal;
31         GameObject pyramidDoor;
32
33
34         public GameObject enemyBoss1;
35         public Transform boss1Respawn;
36
37         public GameObject enemyBoss2;
38         public Transform boss2Respawn;
39
40         GameObject explosionPortal;
41
42         GameObject portalDoor;
43         GameObject player;
44         AudioSource sceneAudio;
45
46         public GameObject infoPlayer;
47
48         public bool allBoxes;
49         private GameObject redPlatform;
50         private GameObject bluePlatform;
51         private GameObject greenPlatform;
52
53         public bool keyInInventary;
54         private bool boxInRed;
55         private bool boxInBlue;
56         public bool boxInGreen;
```

... Game\MultiplayerGame\Assets\Scripts\SceneController.cs 2

```
57          public bool boxesBlueRed;
58          public bool canInstantiateBoss1;
59          public bool canInstantiateBoss2;
60          public bool showText;
61
62          public bool correctWay1;
63          public bool correctWay2;
64          public bool activeAllBoxes;
65          public bool noMoreEnemies1;
66          public bool noMoreEnemies2;
67          public bool openGate;
68          public bool explosionRep;
69          public bool oneKeyInInventary;
70
71          public bool repeatSound1 = true;
72          public bool repeatSound2 = true;
73          public bool repeatSound3 = true;
74
75          List<GameObject> objects = new List<GameObject>();
76
77          void Start()
78          {
79              loadScene = GameObject.Find("LevelLoader");
80              levelLoader = loadScene.GetComponent<LevelLoader>();
81
82              player = GameObject.FindGameObjectWithTag("Player");
83              objects = player.GetComponent<PickUpObjects>().inventary;
84              keysPlayer = player.GetComponent<PickUpObjects>();
85
86
87              cubeBlue = GameObject.Find("BlueCube");
88              cubeGreen = GameObject.Find("GreenCube");
89              cubeRed = GameObject.Find("RedCube");
90              portalFinal = GameObject.Find("PortalFinal");
91              pyramidDoor = GameObject.Find("Pyramid");
92
93              portalDoor = GameObject.Find("PortalDoor");
94              redPlatform = GameObject.Find("PlatformRed");
95              greenPlatform = GameObject.Find("PlatformGreen");
96              bluePlatform = GameObject.Find("PlatformBlue");
97
98              sceneAudio = GetComponent<AudioSource>();
99
100             infoPlayer.SetActive(false);
101
102         }
103
104         // Update is called once per frame
105         void Update()
106         {
107             EnemysRespawn();
108             Controller();
109             SceneAudio();
110
111         }
112
```

... Game\MultiplayerGame\Assets\Scripts\SceneController.cs 3

```
113         public void Controller()
114         {
115             if (player.GetComponent<PickUpObjects>().showPickUpText)
116                 infoPlayer.SetActive(true);
117             else
118                 infoPlayer.SetActive(false);
119
120             if (keysPlayer.getkey1 && keysPlayer.getkey2)
121                 keyInInventary = true;
122
123             if ((keysPlayer.getkey1 && !keysPlayer.getkey2) || (!
                  keysPlayer.getkey1 && keysPlayer.getkey2))
124                 oneKeyInInventary = true;
125
126             boxInRed = redPlatform.GetComponent<PlatformRedController>
                  ().boxInRed;
127             boxInBlue = bluePlatform.GetComponent<PlatformBlueController>
                  ().boxInBlue;
128             boxInGreen = greenPlatform.GetComponent<PlatformGreenController>
                  ().boxInGreen;
129
130             if (boxInRed && boxInBlue && boxInGreen && !explosionRep)
131             {
132                 allBoxes = true;
133             }
134             if (allBoxes)
135             {
136                 explosionRep = true;
137                 allBoxes = false;
138                 GameObject explosion = Instantiate(explosionVFX,
                      explosionPosition.position, Quaternion.identity);
139                 Destroy(explosion, 3.5f);
140                 Destroy(portalFinal);
141
142                 sceneAudio.clip = Resources.Load<AudioClip>("explosionFire");
143                 sceneAudio.Play();
144
145                 Camera.main.GetComponent<CameraShake>().Shake();
146             }
147
148             if (boxInBlue && boxInRed && !noMoreEnemies1)
149             {
150                 canInstantiateBoss1 = true;
151                 boxesBlueRed = true;
152
153             }
154             if (GameObject.Find("Enemy15") == null && !noMoreEnemies2)
155             {
156                 canInstantiateBoss2 = true;
157
158             }
159
160             if (player.GetComponent<PickUpObjects>().activeRed &&
                  player.GetComponent<PickUpObjects>().activeBlue == false &&
                  player.GetComponent<PickUpObjects>().activeGreen == false)
161             {
```

... Game\MultiplayerGame\Assets\Scripts\SceneController.cs 4

```
162                 correctWay1 = true;
163             }
164             if (correctWay1 && player.GetComponent<PickUpObjects>
                  ().activeGreen && !correctWay2)
165             {
166                 correctWay1 = false;
167             }
168 
169             if (player.GetComponent<PickUpObjects>().activeBlue && correctWay1
                   && !correctWay2)
170             {
171                 correctWay2 = true;
172 
173             }
174             if(player.GetComponent<PickUpObjects>().activeGreen && correctWay1
                   && correctWay2)
175             {
176                 openGate = true;
177                 Destroy(cubeBlue, 10);
178                 Destroy(cubeGreen, 10);
179                 Destroy(cubeRed, 10);
180             }
181             if(player.GetComponent<PickUpObjects>().activeRed && correctWay1
                  && correctWay2)
182             {
183                 correctWay1 = false;
184                 correctWay2 = false;
185             }
186         }
187 
188         public void EnemysRespawn()
189         {
190             if (canInstantiateBoss1)
191             {
192                 Instantiate(enemyBoss1, boss1Respawn.transform.position,
                      Quaternion.identity);
193                 canInstantiateBoss1 = false;
194                 noMoreEnemies1 = true;
195             }
196 
197             if (canInstantiateBoss2)
198             {
199                 Instantiate(enemyBoss2, boss2Respawn.transform.position,
                      Quaternion.identity);
200                 canInstantiateBoss2 = false;
201                 noMoreEnemies2 = true;
202 
203                 sceneAudio.clip = Resources.Load<AudioClip>("respawnEnemy");
204                 sceneAudio.Play();
205             }
206 
207         }
208 
209         private void SceneAudio()
210         {
211             if (boxInBlue && repeatSound1)
```

... Game\MultiplayerGame\Assets\Scripts\SceneController.cs 5

```
212             {
213                 sceneAudio.clip = Resources.Load<AudioClip>("getSpecial");
214                 sceneAudio.Play();
215                 repeatSound1 = false;
216 
217             }
218             if (boxInRed && repeatSound2)
219             {
220                 sceneAudio.clip = Resources.Load<AudioClip>("getSpecial");
221                 sceneAudio.Play();
222                 repeatSound2 = false;
223             }
224 
225             if (player.GetComponent<PickUpObjects>().gameFinish)
226             {
227                 youWin.SetActive(true);
228                 timer += Time.deltaTime;
229 
230                 if(timer > timeToEnd)
231                     levelLoader.LoadLevel(3);
232 
233             }
234 
235             if (player.GetComponent<PlayerHealth>().playerDead)
236             {
237                 gameOver.SetActive(true);
238                 timerDead += Time.deltaTime;
239 
240                 if (timerDead > timeToEnd)
241                     SceneManager.LoadScene(0);
242             }
243 
244         }
245 
246     }
247 
248 
249 }
250 
```